Robust Learning in Safety-Related Domains

Machine Learning Methods for Solving
Safety-Related Application Problems

Dissertation

zur Erlangung des akademischen Grades

Doktoringenieur (Dr.-Ing.)

angenommen durch die Fakultät für Informatik
der Otto-von-Guericke-Universität Magdeburg

von Dipl.-Inform. Sebastian Nusser,
geboren am 9. Oktober 1980 in Burg (Magdeburg)

Gutachter:

Prof. Dr. Rudolf Kruse
PD Dr. Christian Borgelt
PD Dr. Thomas Runkler

Magdeburg, 10. Juli 2009

Sebastian Nusser
Robust Learning in Safety-Related Domains – Machine Learning Methods for Solving Safety-Related Application Problems
Dissertation, Otto-von-Guericke-Universität Magdeburg
Magdeburg, 10. Juli 2009

Abstract

Today, machine learning methods are successfully deployed in a wide range of applications. A multitude of different learning algorithms has been developed in order to solve classification and regression problems. These common machine learning approaches are regarded with suspicion by domain experts in safety-related application fields because it is often infeasible to sufficiently interpret and validate the learned solutions. Especially for safety-related applications, it is imperative to guarantee that the learned solution is correct and fulfills all given requirements. The basic idea of the approaches proposed within this thesis is to solve high-dimensional application problems by an ensemble of simple submodels, each of which is allowed to only use two or three dimensions of the complete input space. The restriction of the dimensionality of the submodels allows the visualization of the learned models. Thus a visual interpretation and validation according to the existing domain knowledge becomes feasible. Due to the visualization, an unintended and possibly undesired extra- and interpolation behavior can be discovered and avoided by changing the model parameters or selecting other submodels. Since the learned submodels are interpretable the correctness of the learned solution can therefore be guaranteed. The ensemble of the submodels compensates for the limited dimensionality of the individual submodels. The proposed ensemble methods are successfully applied on common benchmark problems as well as on real-world application problems with very high requirements on the functional safety of the learned solution.

Zusammenfassung

Methoden des Maschinellen Lernens werden heutzutage erfolgreich in vielen Anwendungsgebieten eingesetzt. Eine Vielzahl verschiedener Lernverfahren zur Lösung von Klassifikations- und Regressionsaufgaben existieren bereits. Diese gängigen Methoden des maschinellen Lernens werden von den Domänenexperten im Bereich sicherheitskritischer Systeme mit Skepsis betrachtet, da es oftmals sehr aufwendig ist, die so erzeugten Modelle hinreichend zu interpretieren und zu validieren. Speziell für sicherheitskritische Anwendungen ist es absolut notwendig, dass die Korrektheit und funktionelle Vollständigkeit der gefundenen Lösung garantiert werden kann. Die in dieser Arbeit vorgestellten Lernverfahren erlauben die Interpretation und damit die Validierung der gelernten Modelle durch die Experten. Die Grundidee dieser Methoden besteht darin, hochdimensionale Anwendungsprobleme durch ein Ensemble von einfachen Teilmodellen zu lösen, wobei jedes Teilmodell auf einen nur zwei- oder dreidimensionalen Teilraum des Eingaberaumes beschränkt ist. Diese Beschränkung der Dimensionalität der Teilmodelle ermöglicht die Darstellung der gelernten Modelle. Dadurch wird es möglich, eine visuelle Interpretation und Validierung basierend auf dem existierenden Expertenwissen durchzuführen. Die Visualisierung erlaubt es, ein unerwartetes und möglicherweise unerwünschtes Interpolations- bzw. Extrapolationsverhalten der Teil-

modelle zu entdecken, um dann durch eine entsprechende Änderung der Lernparameter oder eine geänderte Modellauswahl ein solches Verhalten zu vermeiden. Durch diese Vorgehensweise kann die Korrektheit der gelernten Lösung garantiert werden. Das Ensemble der Teilmodelle ermöglicht eine verbesserte Vorhersageleistung im Vergleich zu der eingeschränkten Vorhersageleistung der einzelnen Teilmodelle. Die vorgestellten Lernverfahren liefern sowohl für bekannte Benchmarkdatensätze als auch bei realen Anwendungsproblemen mit sehr hohen Sicherheitsanforderungen gute Ergebnisse.

Contents

1 Introduction

Learning can be seen as the process of extracting *knowledge* out of given *data*. The data usually consists of *observations* of a certain system. The inferred knowledge about this system can be represented by a *model*. For example, someone wants to travel from Munich to Berlin. The railroad timetable lists the following connections leaving Munich at 6:20, at 7:20, at 8:20, at 9:21, at 10:20, at 11:20, at 12:20, at 13:20, at 14:20, at 15:20, at 16:16, at 17:20, and at 18:20. A human typically does not memorize the complete schedule – he generates a shorter description of the system: "Every hour, a train leaves Munich with the destination Berlin – starting at 6:20 and the last connecting train leaves Munich at 18:20." That is, a human is capable to generate a model of the system given a set of observations. In the field of machine learning theory, the term "learning" denotes an automated process of extracting useful information from given observations – e.g. generating rule-like models, discovering patterns, or deriving hypotheses from some given data.

There are numerous machine learning algorithms that are successfully applied to a wide range of applications, for instance, object and speech recognition, search engines, stock and energy market analysis, fraud detection, analysis of medical data, weather forecasting, or even game playing. However, in spite of all these successfully solved application problems, machine learning methods are regarded with suspicion by the domain experts in the field of safety-related applications. Reasons for this skepticism are that the learned models are often hard to verify and that the exact inter- and extrapolation behavior is often unclear. An example of such counterintuitive extrapolation behavior is given in Fig. 1.1, where the prediction of a support vector machine classifier changes within a region not covered by the given data set. Discovering such an unintended and potentially undesired behavior is usually a difficult problem – especially within a high-dimensional input space.

In order to successfully solve a safety-related application problem with a machine learning method it is important to guarantee that the learned solution solves the correct problem and satisfies all functional specifications. These strict requirements are necessary because a safety-related system is a system whose malfunction or failure can lead to serious consequences – for example environmental harm, loss or severe damage of equipment, harm or serious injury of people, or even death. Often, it is impossible to rectify a wrong decision within this domain. Examples of those safety-related application domains are aerospace engineering, automotive industry, medical systems, or process automation.

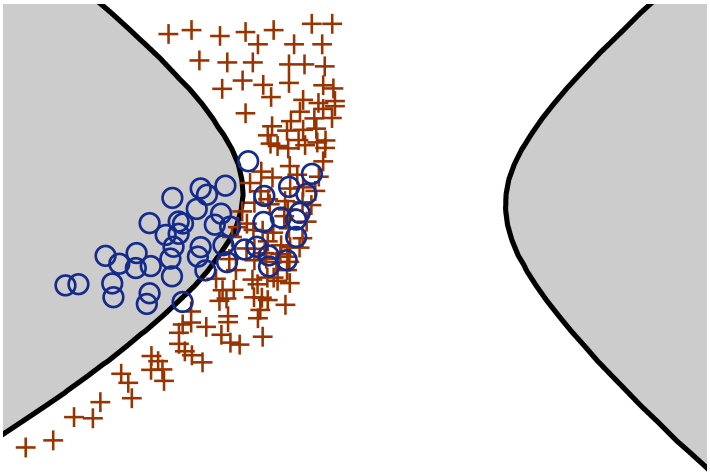


Fig. 1.1: Counterintuitive extrapolation behavior in a region not covered by the given data set. This two-class problem is solved by a support vector machine (SVM) with an acceptable classification performance on the given data. However, in a region not covered by any data the decision of the SVM changes unexpectedly.

1.1 General Motivation

The increasing complexity of safety-related systems and the growth of the number of requirements and customer requests raise the interest of applying machine learning methods within this domain. For instance, the domain knowledge is often imperfect and, for this reason, purely analytical solutions cannot be provided by the domain experts. In addition, the usage of machine learning methods offers a reduction of development time and costs. Furthermore, the predictive performance can be improved by using more sophisticated models. Unfortunately, in the field of safety-related application domains it is often not possible to rectify a wrong decision. Therefore, for effectively applying machine learning methods within this domain, it is crucial to provide strong evidence that the learned solution is valid within the complete input space and correctly satisfies all given functional specifications. That is, it must be ensured that the interpolation and extrapolation behavior of the learned model is correct. Hence, it becomes important to provide an interpretable solution that can be validated according to the given domain knowledge. Since it is infeasible to interpret high-dimensional models sufficiently, these high-dimensional models are usually not applied to safety-related applications. However, simple models which are easier to interpret show a lack of predictive performance.

1.2 Safe Learning for Airbag Control

This thesis is motivated by a challenging real-world problem within the field of automotive safety electronics. It concerns the deployment of restraint systems (e.g.

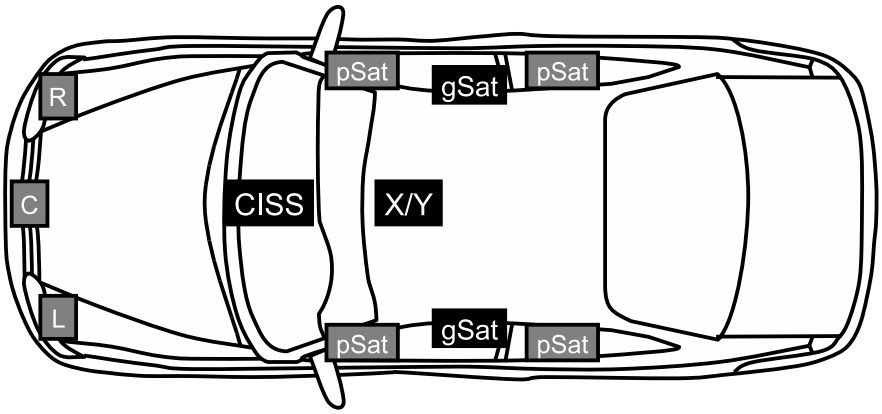


Fig. 1.2: Possible sensor placements in a modern car.

belt pretensioners and airbags) and serves as an example for control applications with high safety requirements. The malfunction of such a system might be fatal and it is impossible to rectify a wrong deployment decision since an airbag system can only be triggered once. Depending on the severity of a crash different restraint systems must be triggered: for instance, the belt pretensioners, the front airbag stage 1 (airbag is inflated to 70%) or stage 2 (airbag is inflated to 100%), the knee airbags, the side airbags (front or rear), or the curtain airbags. Furthermore, the airbag must be triggered within a certain time interval in order to ensure the best passenger protection – a front crash, for example, must be triggered within 15 ms to 30 ms. A late deployment of an airbag can lead to severe injuries of the car passenger.

For each new car platform the control logic of the restraint systems has to be developed from scratch since modifications of mechanical components, different sensor placements, and new functional requirements of a car platform (for example pedestrian protection which may require a softer structure of the bumper) can dramatically influence the signal characteristics and, thus, a solution of the previous platform will not be applicable anymore.

Fig. 1.2 illustrates a possible sensor placement in a modern car. L, C, and R indicate acceleration sensors that are usually placed behind the bumper. X/Y denotes acceleration sensors that are located within the central control unit. The X sensor measures the frontal acceleration and the Y sensor measures the lateral acceleration of the car. gSat refers to satellite sensors that measure the acceleration and pSat denotes satellite sensors measuring the pressure. Another kind of sensor measures the crash impact sound (CISS). Fig. 1.3 shows two sensor signals measured by the X sensor within the central control unit. The sensor signals are very similar for a FIRE crash (the restraint system must be triggered) and a NOFIRE accident (the restraint system must not be triggered) in the time interval before the requested time to deploy the airbag. In this case, distinguishing both crash types based on a single sensor signal is not possible. Usually, the front sensors behind the bumper are used to detect an accident as soon as possible and the sensors within the central control unit are

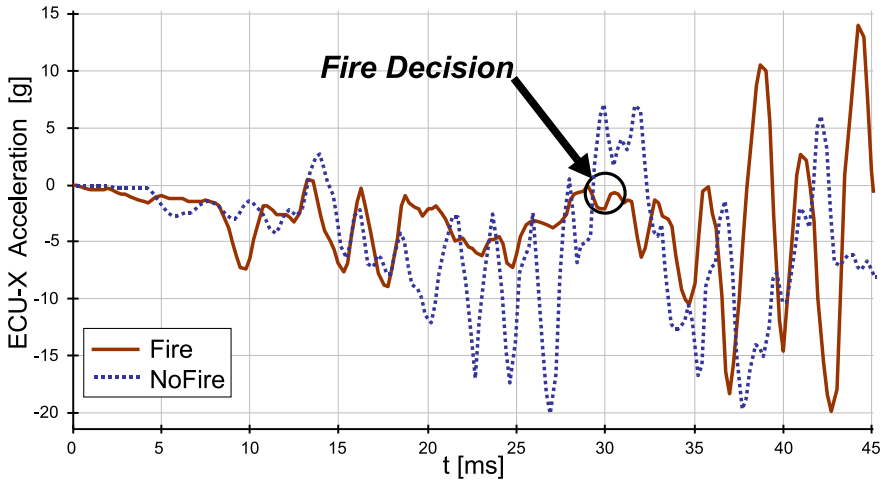


Fig. 1.3: Frontal acceleration measured by the X sensor within the central control unit for two different crash situations. One crash must be triggered, while to second one must not be triggered. The required time of triggering the restraint system is 30 ms. It can be very difficult to distinguish FIRE and NOFIRE events based on a single sensor signal.

used to ensure the robustness of the decision. Further sensors are used in order to enhance the capability of detecting certain crash situations, e.g. side impacts.

Until now, most of the calibration work is done manually by the domain experts. For each crash type different sensor combinations are evaluated and a control logic based on these combinations is developed. This manual calibration is time- and, therefore, cost-intensive. Due to cost pressure in the market the increasing complexity of today's restraint systems must be handled with limited resources. Thus there is a growing interest in automatically learning such a control logic from crash test data in order to reduce development time and costs. A solution of this classification problem is discussed as an application example in Sect. 3.4.1.

1.3 Objective of this Thesis

The objective of this thesis is to provide a machine learning framework that can be used in application domains with very high safety requirements. Therefore, it is necessary to prove that the learned solution is correct, satisfies all given requirements, and complies with the domain knowledge. Sophisticated methods like artificial neural networks (Bishop, 1995; Nabney, 2002) or support vector machines (Schölkopf & Smola, 2002) are known to provide good predictive results, but the interpretation of the fitted model is usually difficult, because each input dimension influences the model in a complex way. On the other hand, simpler methods that usually provide models which show a good interpretability – for instance, linear models, rule-

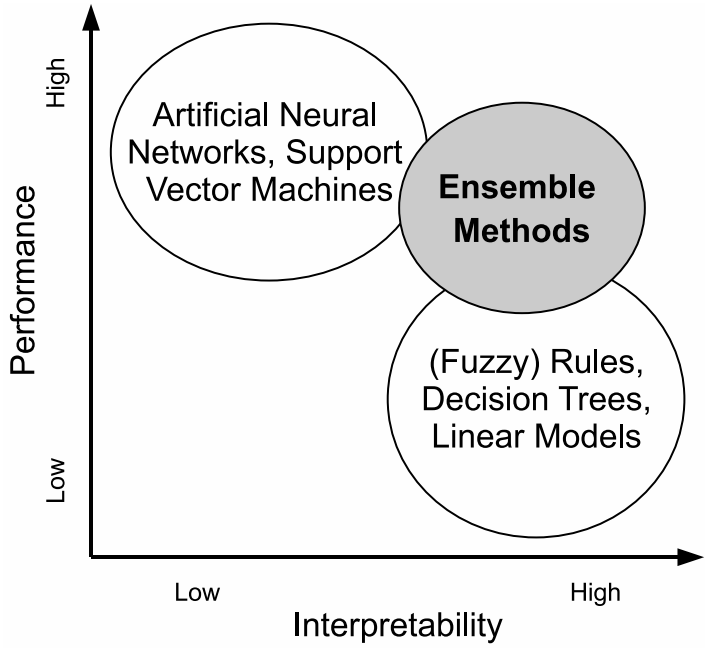


Fig. 1.4: Trade-off between interpretability and predictive performance.

like systems (Lavrac & Dzeroski, 1994; Nauck & Kruse, 1999) and tree-like models (Breiman et al., 1984; Quinlan, 1993) – have a limited predictive performance.

The main focus of interest within this work can be summarized by the following objectives:

1. The data-driven generation of interpretable and verifiable solutions that
2. achieve a good predictive accuracy and
3. allow the inclusion of domain knowledge.

Here, the crucial aspect is to realize a suitable trade-off between (1) and (2). It is obvious that a complex model can achieve a higher predictive performance on the available data. However, a higher model complexity will lead to an increased effort for model verification. This trade-off and the corresponding characterization of common machine learning methods is illustrated in Fig. 1.4. Methods that are known to achieve a high predictive performance – e.g. support vector machines (SVMs) or artificial neural networks (ANNs) – are usually hard to interpret. On the other hand, methods that are known to be well-interpretable – for example (fuzzy) rule systems, decision trees, or linear models – are usually limited with respect to their predictive performance. The use of ensemble methods (Dietterich, 2000; Kuncheva, 2004) can provide an appealing solution of this conflict, because the ensemble modeling approach allows to model smaller subproblems of the original problem and to combine the solutions of the subproblems in order to obtain the final solution. Hereby, it is assumed that the solutions of the subproblems are easier to interpret and their combination remains interpretable.

1.4 Outline

The remaining chapters are organized as follows:

Chap. 2 recalls the basic concepts of machine learning theory. Commonly used machine learning algorithms that are used for comparison within the following chapters are briefly discussed. Furthermore, a brief introduction into the field of safety-related systems is given – including safety-standards, the problem of evaluating the learned solutions of safety-related problems, and existing machine learning approaches that can be used within safety-related domains.

Chap. 3 discusses our classification framework based on ensembles of low-dimensional submodels that is suitable for solving application problems with very high safety requirements. First, we introduce our ensemble framework, which is designed to solve binary classification problems. Two algorithms are developed. The first approach is based on a tree structure where it is necessary to take the decisions of preceding nodes into account. The second approach exploits a typical property of safety-related systems that a certain state of such system should never be misclassified. By exploiting this property, one yields an ensemble model where each submodel can be interpreted independently. Later within this chapter, we discuss different methods to extend our binary classification framework in order to deal with multi-class problems. Finally, we illustrate the advantages of our classification method by two real-world application problems.

Chap. 4 presents a regression algorithm that provides a good trade-off between interpretability and predictive performance of the learned solution. This method uses linear submodels in order to build a piecewise linear regression model. The submodels are determined by an EM-like approach that incorporates information about the target variable in order to obtain a meaningful partitioning of the input space. The proposed algorithm allows the reduction of the dimensionality of the submodels as well as a cluster pruning strategy which is useful for problems where the number of clusters is unknown.

Chap. 5 is concerned with the problem of feature extraction, which is an important issue for successfully applying the algorithms given in Chap. 3 and Chap. 4. The first section addresses the problem of feature selection. Three feature selection methods are discussed and evaluated on several benchmark data sets. The second section relates to feature construction which can become necessary for problems which require a higher dimensionality of the submodels. In addition, data filtering methods are discussed allowing to remove possible conflicts within the training data.

Chap. 6 concludes and summarizes the main contributions of this thesis. Moreover, perspectives of further research are pointed out.

Appendix A summarizes further experiments performed on common benchmark data sets applying our classification and regression algorithms.

Appendix B gives a brief introduction into the analysis of ROC curves.

Appendix C serves as short summary of the notation used within this thesis.

1.5 Publications

Parts of this thesis have been published in Nusser et al. (2007, 2008a,b,c, 2009a,b,c,d); Otte et al. (2006).

2 Machine Learning and Safety-Related Domains

This chapter gives a brief introduction into the field of machine learning theory and safety-related domains. The first matter is discussed in the first section and the second section gives a short introduction into the field of safety-related applications and summarizes recent approaches to use machine learning methods within this domain.

2.1 Learning From Data

Within this section, we discuss different machine learning methods for regression and classification problems that are commonly used. These methods serve as comparison to our own methods which are described in Chap. 3 and Chap. 4. For a more detailed discussion about machine learning theory and machine learning algorithms see, for instance, Bishop (2006); Duda et al. (2001); Hastie et al. (2001); Mitchell (1997). A good introduction into probability theory and statistics can be found, for example, in Milton & Arnold (2002).

2.1.1 Fundamentals of Machine Learning

As already mentioned in Chap. 1, learning is the process of extracting knowledge about a system from observed data. The system is defined by a set of *random variables*[1]. Such random variables are denoted as uppercase letters $X, Y, X_1, X_2, \ldots, X_n, \ldots, X_N$. The N-dimensional *input space* of a system is denoted as $V^N = X_1 \times X_2 \times \ldots \times X_N = \times_{n=1}^{N} X_n$, where X_n denotes the n-th input variable. The *target variable* Y denotes the attribute of a system that should be predicted. The *observed values* of the random variables are denoted as lowercase letters x, y. An observed data point within the input space V^N is denoted as $\vec{v} \in V^N$. The observed data is collected in the *data set* $\mathfrak{D} = \{(\vec{v}_1, y_1), \ldots, (\vec{v}_m, y_m), \ldots, (\vec{v}_M, y_M)\}$, where $\vec{v}_m$ denotes the m-th observation within the input space V^N and y_m denotes the corresponding value of the

[1] Random variables are also denoted as attributes or features.

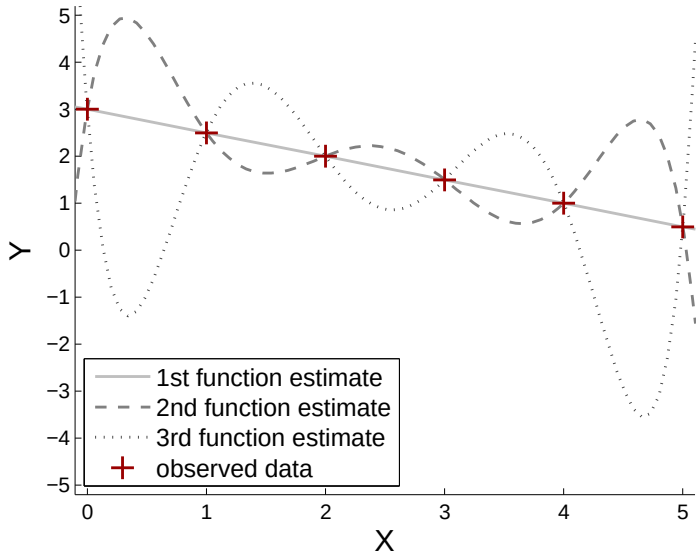


Fig. 2.1: A simple regression problem and three possible regression models.

target variable Y. The underlying process that generates the observations is called the *true target function*

$$f : V^N \rightarrow Y, \tag{2.1}$$

which maps the input space V^N to the target variable Y. Usually, this process is unknown. The learning task is to determine a *function estimate* that maps the observed data points to the corresponding target values, that is

$$\hat{f}(\vec{v}_m) = y_m . \tag{2.2}$$

2.1.2 Regression

A regression task concerns the problem of determining an estimate of functional relationship between the input space $V^N = \times_{n=1}^{N} X_n$, where $X_n \subseteq \mathbb{R}$, and the real-valued target variable $Y \subseteq \mathbb{R}$, that is the task is to find an estimate of the unknown function

$$f : V^N \rightarrow \mathbb{R} \tag{2.3}$$

given the observed data.

Fig. 2.1 illustrates a simple regression problem and three possible solution of this problem. The first function estimate assumes a linear relationship between X and Y and the other two function estimates assume polynomial functions. Within this thesis (cf. Chap. 3 and Chap. 4), we prefer a simple solution of the given problem in order to allow a good interpretability of the models. This is related to the so-called *Occam's razor* (Thorburn, 1918) that advises to use the simplest model fitting the observed data. The situation in which a model perfectly fits the observed data

but has a higher model complexity than necessary is called *overfitting*. The second and third function estimate of Fig. 2.1 are examples of overfitting.

The simplest regression setting assumes a linear relationship among the variables. In this setting, the *expected value* of the target variable Y given the input variables takes

$$\begin{aligned}\hat{f}\left(V^N\right) &= \mathbb{E}\left[Y|X_1,\ldots,X_n,\ldots,X_N\right] \\ &= \alpha_0 + X_1\alpha_1 + \ldots + X_n\alpha_n + \ldots + X_N\alpha_N\,,\end{aligned} \tag{2.4}$$

where α_0 is the *bias* and α_n are the unknown parameters of the *linear regression model*. The regression task is to determine an appropriate estimate of the parameter vector $\vec{\alpha} = (\alpha_0, \alpha_1, \ldots, \alpha_N)^T$. This problem can be solved by

$$\vec{\alpha} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\vec{y}\,, \tag{2.5}$$

where $\vec{y} = (y_1, y_2, \ldots, y_m, \ldots, y_M)^T$ is a vector that stores all observed values of the target variable Y and the observations of all input variables are stored in the M-by-$(N+1)$ matrix $\mathbf{X}$,

$$\mathbf{X} = \begin{bmatrix} 1 & x_1^{(1)} & x_1^{(2)} & \ldots & x_1^{(n)} & \ldots & x_1^{(N)} \\ 1 & x_2^{(1)} & x_2^{(2)} & \ldots & x_2^{(n)} & \ldots & x_2^{(N)} \\ \vdots & \vdots & \vdots & & \vdots & & \vdots \\ 1 & x_m^{(1)} & x_m^{(2)} & \ldots & x_m^{(n)} & \ldots & x_m^{(N)} \\ \vdots & \vdots & \vdots & & \vdots & & \vdots \\ 1 & x_M^{(1)} & x_M^{(2)} & \ldots & x_M^{(n)} & \ldots & x_M^{(N)} \end{bmatrix},$$

where $x_m^{(n)}$ is the m-th observation of random variable X_n. For a detailed discussion of the derivation of Eq. 2.5 see, for instance, (Milton & Arnold, 2002, Chapter 12.2).

For the example data set given in Fig. 2.1, the matrix $\mathbf{X}$ is

$$\mathbf{X} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \\ 1 & 5 \end{bmatrix}$$

and the observed target values are $\vec{y} = (3.0, 2.5, 2.0, 1.5, 1.0, 0.5)^T$. Using Eq. 2.5, the estimated parameter vector of the linear regression model is $\vec{\alpha} = (3, -0.5)^T$ and, thus, the estimated function is

$$\hat{f}(x) = 3 - 0.5x\,.$$

The linear regression model can be extended to a *polynomial regression model* by generating new variables that are basis expansions of the original variables, for example, $X_2 = X_1^2$ and $X_3 = X_1^3$.

Another possible extension of the linear regression model is to replace the linear coefficients α_n by arbitrary functions $f_n(\cdot)$. That is, one obtains a *generalized additive model* (Hastie & Tibshirani, 1990; Stone, 1985):

$$\begin{aligned}\hat{f}\left(V^N\right) &= \mathbb{E}\left[Y|X_1,\ldots,X_n,\ldots,X_N\right] \\ &= f_1\left(X_1\right)+\ldots+f_n\left(X_n\right)+\ldots+f_N\left(X_N\right),\end{aligned} \tag{2.6}$$

which was one of the motivations of developing the methods described in Chap. 3. For details on fitting such a generalized additive model, see (Hastie et al., 2001, Chapter 9.1).

2.1.3 Classification

In a classification setting, one is interested in determining an estimate of the unknown functional relationship between the input space V^N and the target variable Y that can take discrete values out of $\mathbb{K} = \{c_1, c_2, \ldots, c_k, \ldots, c_K\}$. Hence, a *classification model* is a function

$$f : V^N \rightarrow \mathbb{K}. \tag{2.7}$$

The simplest kind of classification problems is a *binary classification* problem, that is, the task is to discriminate two classes, typically labeled as $\mathbb{K} = \{-1, 1\}$. Such classification problem can be solved by regarding the problem as regression problem and to use the sign of the prediction as target value. The resulting decision border is called *separating hyperplane*:

$$0 = \alpha_0 + X_1\alpha_1 + X_2\alpha_2 + \ldots + X_N\alpha_N \tag{2.8}$$

and the *decision function* is:

$$\hat{f}\left(V^N\right) = \operatorname{sign}\left(\alpha_0 + X_1\alpha_1 + X_2\alpha_2 + \ldots + X_N\alpha_N\right). \tag{2.9}$$

That is a new data point $\vec{v}$ is assigned to class -1 or class 1 according to:

$$\hat{f}\left(\vec{v}\right) = \operatorname{sign}\left(\vec{v}^T\vec{w} + \alpha_0\right), \tag{2.10}$$

where $\vec{w} = (\alpha_1, \alpha_2, \ldots, \alpha_N)^T$.

Fig. 2.2 illustrates a possible separating hyperplane for a two-dimensional binary classification problem. The parameter vector $\vec{\alpha}$ is determined according to Eq. 2.5 and the resulting parameter vector $\vec{\alpha}$ is put into Eq. 2.8. The solid line represents the resulting decision border of the estimated classification model. Unfortunately, infinitely many separating hyperplanes are possible.

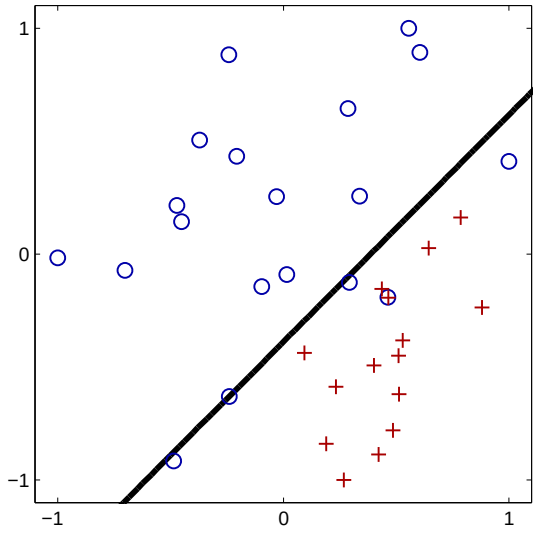


Fig. 2.2: Linear regression used for binary classification. Samples that belong to class -1 are depicted as ◯ and samples that belong to class 1 are depicted as $+$. The separating hyperplane is drawn as solid line.

2.1.4 Support Vector Machines

A support vector machine (SVM) (Boser et al., 1992; Cortes & Vapnik, 1995) determines the *optimal separating hyperplane* between two classes in order to maximize the distance to the closest point from either class. The advantages of this approach are that a unique solution can be provided and a better classification performance on previously unseen data can be achieved.

The SVM constructs two parallel separating hyperplanes $\vec{w}^T\vec{w} + \alpha_0 \geqslant 1$ and $\vec{w}^T\vec{w} + \alpha_0 \leqslant -1$, where $\vec{w} = (\alpha_1, \alpha_2, \ldots, \alpha_N)^T$. In order to obtain the optimal separating hyperplane the distance (also called the margin) between both hyperplanes has to be maximized, $2/\|\vec{w}\|$. This corresponds to the following optimization problem:

$$\begin{aligned} & \min_{\vec{w},\alpha_0} \frac{1}{2}\vec{w}^T\vec{w} \\ \text{subject to} \quad & \vec{v}_m^T\vec{w} + \alpha_0 \geqslant \quad 1 \quad \text{if} \quad y_m = \quad 1 \\ & \vec{v}_m^T\vec{w} + \alpha_0 \leqslant -1 \quad \text{if} \quad y_m = -1 \\ \text{where} \quad & m = 1, \ldots, M\,. \end{aligned} \tag{2.11}$$

The constraints can be simplified to $y_m\left(\vec{v}_m^T\vec{w} + \alpha_0\right) \geqslant 1$. This formulation assumes that the classes are separable within the feature space. By introducing the slack variable $\vec{\xi} = (\xi_1, \xi_2, \ldots, \xi_M)^T$ with $\xi_m \geqslant 0$ and by modifying the constraint in Eq. 2.11 to

$y_m \left(\vec{v}_m^T \vec{w} + \alpha_0\right) \geqslant 1 - \xi_m$ it becomes feasible to deal with overlapping classes within the feature space:

$$\begin{aligned} & \min_{\vec{w},\alpha_0,\vec{\xi}} \frac{1}{2}\vec{w}^T\vec{w} + C\sum_{m=1}^{M}\xi_m \\ \text{subject to} \quad & y_m\left(\vec{v}_m^T\vec{w} + \alpha_0\right) \geqslant 1 - \xi_m, \\ & \xi_m \geqslant 0, m = 1,\ldots,M\,, \end{aligned} \tag{2.12}$$

where $C > 0$ is a constant that allows to control the influence of the outliers (the overlapping data points).

In order to allow also nonlinear functions, one can used the so-called *kernel trick* (Aizerman et al., 1964). That is, the training vectors $\vec{v}_m$ are mapped into a higher-dimensional space $\mathcal{H}$ by the function $\phi : V^N \to \mathcal{H}$. Then the SVM determines the linear separating hyperplane with the maximal margin within this higher dimensional space $\mathcal{H}$. This yields the final optimization problem:

$$\begin{aligned} & \min_{\vec{w}_{\mathcal{H}},\alpha_0,\vec{\xi}} \frac{1}{2}\vec{w}_{\mathcal{H}}^T\vec{w}_{\mathcal{H}} + C\sum_{m=1}^{M}\xi_m \\ \text{subject to} \quad & y_m\left(\phi\left(\vec{v}_m\right)^T\vec{w}_{\mathcal{H}} + \alpha_0\right) \geqslant 1 - \xi_m, \\ & \xi_m \geqslant 0, m = 1,\ldots,M\,, \end{aligned} \tag{2.13}$$

where $\vec{w}_{\mathcal{H}} \in \mathcal{H}$ and the original dot product $\vec{v}_i^T\vec{v}_j$ is replaced by the kernel function $\kappa(\vec{v}_i,\vec{v}_j) = \phi(\vec{v}_i)^T\phi(\vec{v}_j)$. Examples for such kernel functions are

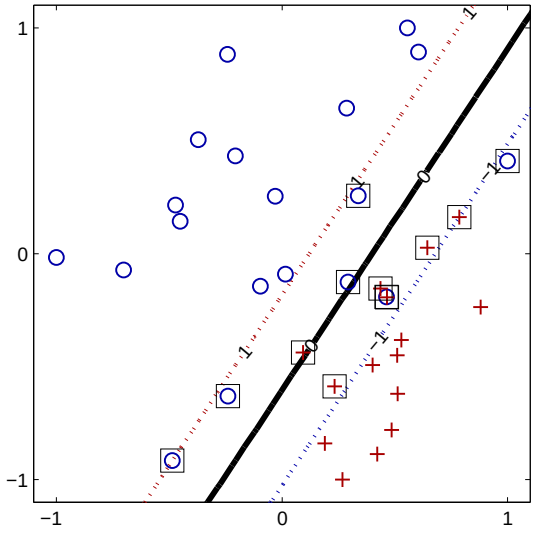
the linear kernel $\kappa\left(\vec{v}_i,\vec{v}_j\right) = \vec{v}_i^T\vec{v}_j$ (an SVM with a linear kernel is shown in Fig. 2.3(a)) and

the Gaussian kernel $\kappa\left(\vec{v}_i,\vec{v}_j\right) = \exp\left(-\gamma\left\|\vec{v}_i - \vec{v}_j\right\|^2\right)$, where $\gamma > 0$ is the kernel parameter which controls the spread of the Gaussian (an example of an SVM with a Gaussian kernel is shown in Fig. 2.3(b)).

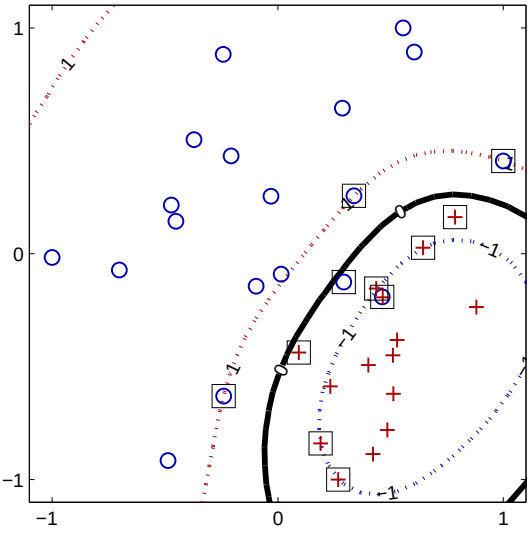
The final SVM decision function is

$$\hat{f}\left(\vec{v}\right) = \operatorname{sign}\left(\phi(\vec{v})^T\vec{w}_{\mathcal{H}} + \alpha_0\right)\,. \tag{2.14}$$

In contrast to Eq. 2.9, this decision function is not necessarily linear within the input space. The kernel trick can increase the predictive performance of non-linear classification (cf. Fig. 2.3) and provides a powerful framework for solving a multitude of classification problems. However, the non-linear transformation decreases the interpretability of the learned solution – at least for non-experts in machine learning. A very good and more detailed discussion of support vector machines and further kernel-based approaches can be found in Schölkopf & Smola (2002).



(a) Support vector machine with a linear kernel function.



(b) Support vector machine with a Gaussian kernel function.

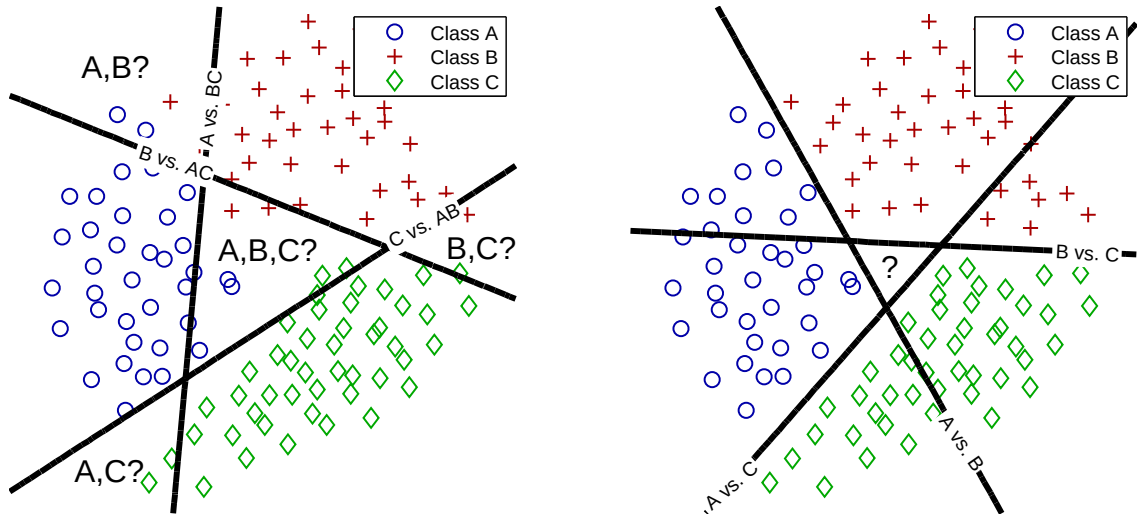
Fig. 2.3: Two support vector machines learned on the binary classification problem from Fig. 2.2. The solid line represents the decision border of each classifier and the dotted lines indicate the margins, that is $f(x) = \pm 1$. The support vectors (that is the data points that are within the margin) are surrounded by black squares.

2.1.5 Multi-Class Extensions of Binary Classifiers

The support vector machine (cf. Sect. 2.1.4) provides an elegant solution for binary classification problems. Unfortunately, there are numerous classification problems that concern more than two classes. Thus it is necessary to find an adequate extension of the binary classifier. Two commonly used approaches of extending binary classifiers in order to solve multi-class problems are: (1) the one-against-one extension and (2) the one-against-rest extension. A detailed comparison of these methods and an experimental evaluation for support vector machines is given in Hsu & Lin (2002). Fig. 2.4 illustrates both approaches.

One-against-rest multi-class extension. This method constructs K classifiers, where $K = |\mathbb{K}|$ is the number of classes. The model f_{c_k} for class $c_k \in \mathbb{K}$ is trained on all samples of class c_k against all samples from the remaining classes which are combined to a new class $c_k^* = \mathbb{K} \setminus c_k$, for the sake of simplicity the class label of c_k^* is set to -1. A new data point $\vec{v}$ is assigned to: $f(\vec{v}) = \arg\max_{c_k \in \mathbb{K}} f_{c_k}(\vec{v})$.

One-against-one multi-class extension. This method builds $K(K-1)/2$ binary classification models, each for the pair-wise combination of the classes $c_k, c_l \in \mathbb{K}$, $k \neq l$.



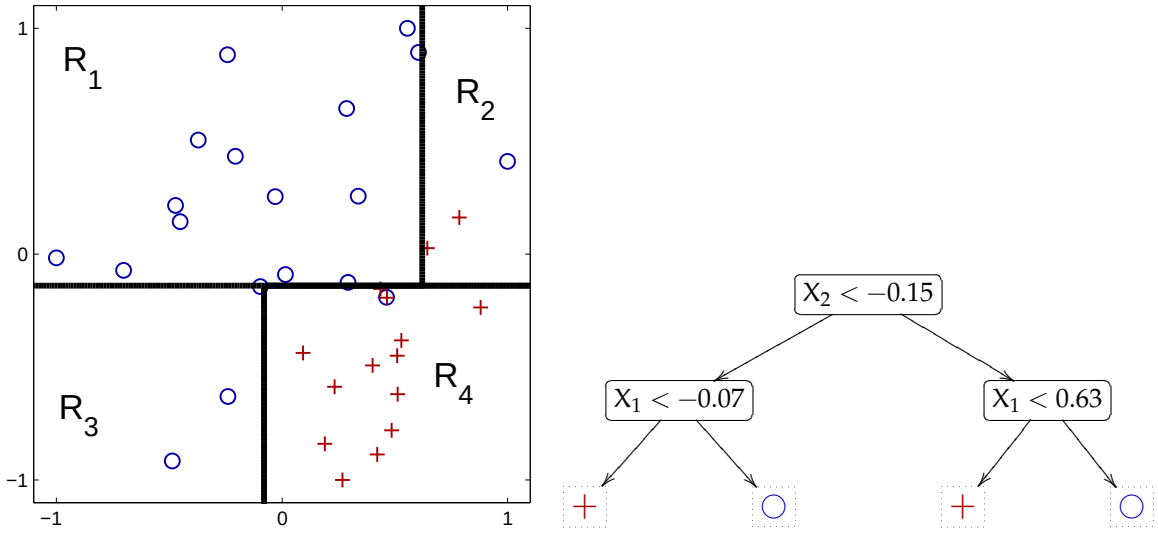
(a) One-against-rest extension.

(b) One-against-one extension.

Fig. 2.4: Illustration of multi-class extensions based on binary classifiers. There are three classes: A, B, C. The discriminant functions are given as solid lines. Regions with possible inconsistent decisions are labeled with question marks.

The final classification is performed by majority voting – that is the most frequent predicted class label is returned as prediction of the multi-class model.

Risk of inconsistent decisions. The issue of inconsistent decisions of combining binary classifiers to multi-class classifiers is addressed, e.g., in Tax & Duin (2002). As illustrated in Fig. 2.4, there can be regions of the input space where the decision of the multi-class models might be inconsistent. Those regions are marked with question marks in each figure. For the *one-against-rest method*, there are two possibilities of an inconsistent decision: (1) there are several binary classifiers predicting different class labels for one given data point. Such regions are (A,B ?), (A,C ?), (B,C ?). (2) there are regions, where all classifiers are predicting the "rest" class, (A,B,C ?). For the *one-against-one method*, there is only one kind of inconsistent decisions possible: several binary classifiers are predicting different class label for one given data point. The problem of several classifiers predicting different class labels can be solved by assigning the class label at random (Hsu & Lin, 2002) or to assign the data point to the class with the highest posterior probability (Tax & Duin, 2002). The second kind of inconsistent decisions of the one-against-rest method can be acceptable for some problems, where "no decision" might be better than a "wrong decision". Otherwise, one can use the same strategy as for the other kind of inconsistent decisions.



(a) Partitioning of the input space into four regions.

(b) Corresponding classification tree model.

Fig. 2.5: Classification tree learned on a simple binary classification problem.

2.1.6 Classification and Regression Trees

In contrast to the methods described above a tree-based method does not try to solve the given learning problem by one single model. Tree-based methods follow a divide-and-conquer strategy, where a complex problem is tackled by dividing it recursively into simpler subproblems. This strategy yields a recursive partitioning of the input space into a set of hyper-rectangles. For each of those rectangles within the input space a different model is fitted. In this section, we will focus on the CART method (Breiman et al., 1984) – other tree learning algorithms like C4.5 (Quinlan, 1993) or AID (Morgan & Sonquist, 1963) are based on similar principles and, thus, they are omitted within this introduction. Extensions of the original tree-based models that are not restricted to axes-parallel hyperplanes are given, for instance, in Brodley & Utgoff (1995); Gama (2000); Murthy et al. (1994).

Fig. 2.5(a) illustrates a recursive partitioning of the input space which is determined on the data set from Fig. 2.2. The first binary split is $X_2 < -0.15$, which divides the input space into two subregions $R_1 \cup R_2$ and $R_3 \cup R_4$. Both subregions can be subdivided independently. That is, subregion $R_1 \cup R_2$ is divided by $X_1 < 0.63$ into R_1 and R_2 and subregion $R_3 \cup R_4$ is divided by $X_1 < -0.07$ into R_3 and R_4, respectively. This recursive partitioning of the input space can be represented by a graphical model as depicted in Fig. 2.5(b). The inner nodes of the tree are called *decision nodes* and the end nodes are called *leaf nodes*. The node at the top of the tree is called the root node.

For any new data point $\vec{v}$, it is determined in which region it falls into by starting at the root of the tree and following the path down to a leaf node according to the decisions made by the inner nodes. Each leaf node of the tree encodes a certain

model prediction. Within a classification setting the prediction of the leaf node is a certain class label (as shown in Fig. 2.5(b)) and within a regression setting this prediction is a constant value (as illustrated in Fig. 2.6(a)).

The final function estimate can be written as:

$$\hat{f}(\vec{v}) = \sum_{\tau=1}^{T} t_\tau I(\vec{v} \in R_\tau)\,, \tag{2.15}$$

where $R_1, R_2, \ldots, R_\tau, \ldots, R_T$ are the regions of the input space V^N which are defined by the binary splits of the tree, t_τ is the prediction of the leaf node that corresponds to region R_τ, and $I(\cdot)$ is the indicator function, $I(A) = \begin{cases} 1 & \text{if} \quad A \text{ is true} \\ 0 & \text{if} \quad A \text{ is false} \end{cases}$.

Classification Trees. The prediction for region R_τ of a classification tree is the majority class within this region. That is, the prediction for region R_τ is

$$t_\tau = \arg\max_{c_k} p_{\tau,k}\,, \tag{2.16}$$

where

$$p_{\tau,k} = \frac{1}{M_\tau} \sum_{\vec{v}_m \in R_\tau} I(y_m = c_k) \tag{2.17}$$

is the proportion of observations of class c_k within region R_τ and M_τ is the number of all data points within R_τ.

The best split for a node τ in a classification tree (a region R_τ) is determined by an impurity function (Breiman, 1996) $i(\tau, \vec{p})$, where $\vec{p} = (p_1, p_2, \ldots, p_k, \ldots, p_K)^T$ are the proportions within node τ, that is, the proportion of every class c_k in the current node τ. One commonly used impurity measure is the Gini function:

$$i_{\text{Gini}}(\tau, \vec{p}) = \sum_{k=1}^{K} p_k(1 - p_k)\,. \tag{2.18}$$

Another often used impurity measure is the entropy

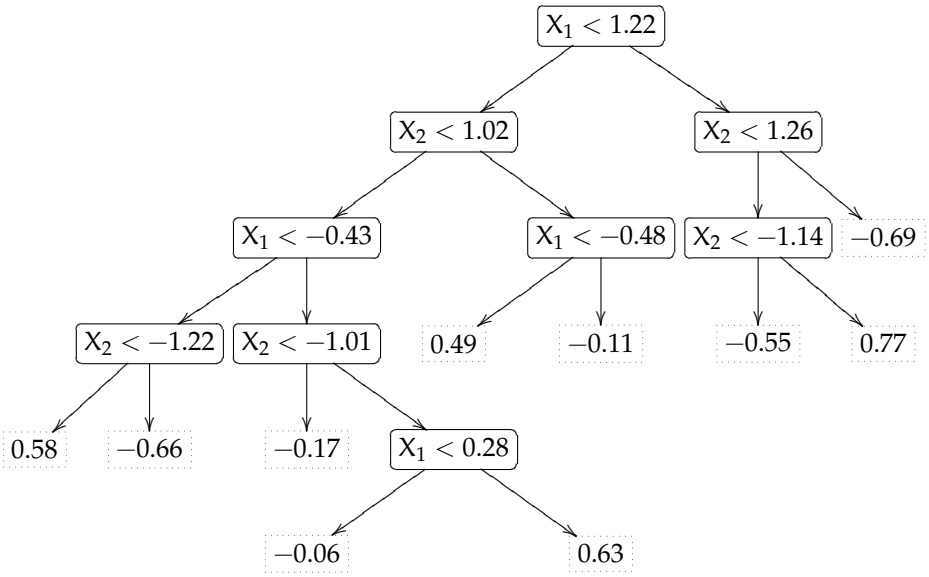
$$i_{\text{Entropy}}(\tau, \vec{p}) = -\sum_{k=1}^{K} p_k \log_2 p_k\,. \tag{2.19}$$

Choosing one of both impurity measures, the goodness-of-split criterion[2] is

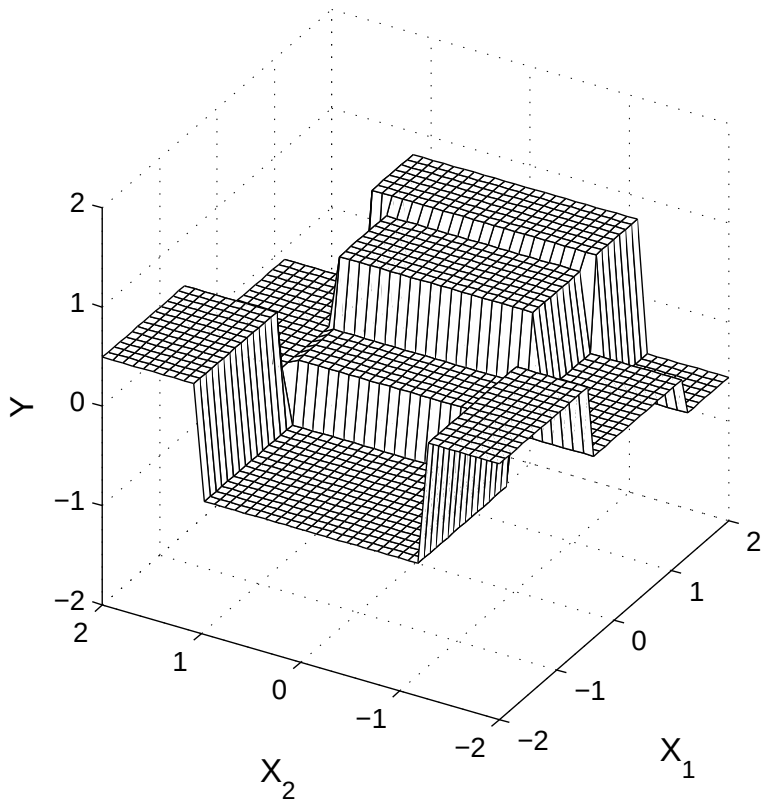
$$\Delta i(\tau) = i(\tau, \vec{p}) - P_L i(\tau_L, \vec{p}_L) - P_R i(\tau_R, \vec{p}_R)\,, \tag{2.20}$$

where τ_L and τ_R are the left and right descendent nodes of node τ, $i(\tau_L, \vec{p}_L)$ and $i(\tau_R, \vec{p}_R)$ are their impurity measures, and P_L and $P_R = 1 - P_L$ are the fractions of data points assigned to node τ_L and τ_R, respectively. The best split of the node τ maximizes Eq. 2.20.

[2] A detailed discussion of different selection measures is given, for instance, in Borgelt & Kruse (1998).



(a) Regression tree model.



(b) Regression tree prediction.

Fig. 2.6: Regression tree learned on an artificial regression problem.

Regression Trees. The prediction for region R_τ defined by leaf node τ of a regression tree is

$$t_\tau = \frac{1}{M_\tau} \sum_{\vec{v}_m \in R_\tau} y_m \,, \tag{2.21}$$

where M_τ is the number of all data points within region R_τ.

The best split within the regression setting is determined by

$$\min_{(n,\theta)} \left[\min_{t_L} \sum_{x_m^{(n)} < \theta} (y_m - t_L)^2 + \min_{t_R} \sum_{x_m^{(n)} \geqslant \theta} (y_m - t_R)^2 \right] , \tag{2.22}$$

where θ is the threshold of the binary split.

Tree-based methods are simple but powerful. Although, the graphical representation is easy to understand, a tree-based model can become quite complex – as illustrated in Fig. 2.6, where a regression tree is learned for a simple toy example. Nevertheless, tree-based models can be regarded as well-interpretable compared to other methods.

2.1.7 Model Selection

As already mentioned in Sect. 2.1.2, we are interested in a simple solution that should provide a good performance on the given problem. Since the true target function is usually unknown, it can become difficult to decide which function estimate is the best solution. The predictive error of a model can be used as evaluation measure for choosing the best model. In a regression setting the predictive error can be estimated by

$$\widehat{err} = \frac{1}{M} \sum_{m=1}^{M} \left| y_m - \hat{f}(\vec{v}_m) \right| \tag{2.23}$$

and for a classification problem the predictive error can be estimated by

$$\widehat{err} = \frac{1}{M} \sum_{m=1}^{M} I\left(y_m \neq \hat{f}\left(\vec{v}_m\right) \right) \,. \tag{2.24}$$

Evaluating a learned model solely on the data used for training is not advisable, since it is not possible to make statements about the ability of the learned solution to correctly predict the target values of previously unseen data points. This ability is called *generalization performance*. A learning algorithm can perfectly predict the target value for all data points within the set of observations but on novel data points where the target value is unknown it might achieve a poor predictive performance. This problem is illustrated in Fig. 2.1, where f_2 and f_3 overfit the observed data points.

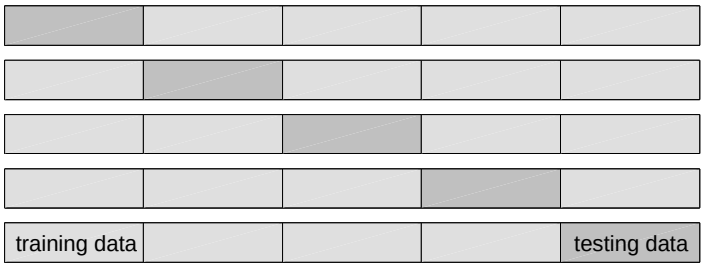


Fig. 2.7: Illustration of a five-fold crossvalidation procedure. The data set is divided into five roughly equal-sized folds. Each fold is used once as testing data set, while the remaining folds are used for training. The error estimate based on the testing data sets is averaged over all runs.

In order to evaluate the solution generated by a certain learning method, one can divide the data set into a training data set and a testing data set. The training data is used for building the model and the testing data is only used to estimate the predictive performance of the model. This yields an unbiased estimated of the generalization performance. In order to exploit all available data and to obtain a more robust estimate of the generalization performance of a learning method one can perform a so-called K-*fold crossvalidation* procedure (Kohavi, 1995a). This procedure divides the data set into K subsets which are called the folds. Each fold is used once as testing data set while the remaining folds are used for model training. Thus this method generates K models. The generalization performance of the learning method used for model building is estimated by averaging the predictive performance of each learned model on the testing data sets. K-fold crossvalidation is used in Appendix A in order to compare our learning algorithms of Chap. 3 and Chap. 4 with the commonly used learning algorithms which are given in this chapter.

2.2 Safety-Related Systems

Safety is "a measure of the continuous delivery of a service free from occurrences of catastrophic failures" (Bowen & Hinchey, 1999). That is, safety is the property of a system that it will not endanger human life or the environment. Therefore, a safety-critical system is a system where a failure may result in injury or loss of life[3]. Systems that involve safety aspects but are not necessarily safety-critical are so-called safety-related systems. Safety-critical systems can be seen as a subset of safety-related systems. Within this thesis, we will not distinguish between safety-critical and safety-related systems.

[3] An often quoted example of the malfunctioning of such a safety-critical system is the Therac-25 radiotherapy machine which killed several people (Leveson & Turner, 1993).

Machine learning methods in the domain of safety-related applications can be grouped into three classes where the safety requirements of each class are increasing:

Decision Support Systems. This application class includes forecasts, planning supports, recommendations of control variables and decision support systems in medical engineering or system control. These systems take on a purely advisory role for the domain experts or the plant operators.

Monitoring and Diagnosis. Monitoring systems for engines, equipment, production or process facilities, and systems for fault diagnosis and early fault detection can be assigned to this application class. Typically, an alarm is activated in such systems leading to a subsequent user intervention. A wrong decision of such a monitoring system (for instance a false alarm) can be rectified by the domain expert or system operator.

Automation and Control. This class of applications is most challenging since the systems perform autonomously – without any user interaction – and directly affect the control or automation task. This application class has particularly high requirements on the functional safety and it is especially important to satisfy the domain experts' demand for the correctness of the learned solution.

The machine learning approaches discussed in Sect. 3.2 and Sect. 3.3 are developed for autonomous control tasks – but they are also applicable for the application classes with lower safety-requirements.

2.2.1 Safety Standards

Safety-related systems can be found in many application domains, for instance in industry (manufacturing control and robots), transportation (systems on-board aircraft, air-traffic management, train control systems, automotive safety electronics), communication (dispatch systems and emergency-call systems), and medicine (medical monitoring, medical robots, and radiation therapy systems) – see Isaksen et al. (1997). Thus there are numerous standards that concern the analysis of safety-related systems and safety-related software, for instance DEFSTAN 00-56 (2007); IEC-61508 (2005); MISRA (1994). The IEC-61508 standard is primarily originated by process and automation industries. It can be used as a standalone standard or as a basis for other sector- or product-specific standards, for instance EN-5012x (railways), IEC-60601 (medical), IEC-62021 (machinery), or IEC-61513 (nuclear). There are also several approaches to establish a standard – or at least a guideline – for certifying machine learning methods (for instance artificial neural networks) for usage in safety-related application domains, see Bedford et al. (1996); Pullum et al. (2007); Taylor (2005).

Safety deals with the protection against hazards and risks that can arise from the operation of a safety-related system. Several standards concern the problem of defining safety integrity levels (SILs), for instance IEC-61508 (2005) and MISRA (1994),

Tab. 2.1: Safety Integrity Level (SIL) and corresponding hazard rates from IEC-61508 (2005).

SIL	Mode of Operation	
	Low-Demand	High-Demand
4	$\geqslant 10^{-5}$ to 10^{-4}	$\geqslant 10^{-9}$ to 10^{-8}
3	$\geqslant 10^{-4}$ to 10^{-3}	$\geqslant 10^{-8}$ to 10^{-7}
2	$\geqslant 10^{-3}$ to 10^{-2}	$\geqslant 10^{-7}$ to 10^{-6}
1	$\geqslant 10^{-2}$ to 10^{-1}	$\geqslant 10^{-6}$ to 10^{-5}

where safety integrity states the safety-related system's probability of correctly performing the required safety functions under the given requirements within a certain time period. To define such a hierarchy it is necessary to classify the hazards according to their severity, after which the system can be assigned a certain safety integrity level (SIL). IEC-61508 (2005) differentiates between two so-called modes of operation: The *demand* mode of operation concerns a safety function that is only performed when required in order to transfer the system into another state. The *continuous* safety function is used to retain a system within its normal safe state. The *low-demand* mode of operation is related to the average probability of failure to perform the design function of a safety-related system on demand. This corresponds to the product of the demand rate and the average probability of failure per demand. The *high-demand* and continuous mode of operation concerns the probability of a dangerous failure per hour for the system. Tab. 2.1 relates the safety integrity levels to the required hazard rates for each mode of operation.

2.2.2 Assessing Solutions for Safety-Related Problems

The correctness and reliability of a system is usually assessed by formal verification and validation methods. Thereby, verification concerns the question whether the system is being built right. It is evaluated whether or not the system complies with the imposed specifications and conditions. On the other hand, validation concerns the question whether the right system for the user's needs is being built. This problem is related to statements on system reliability and failure rates, which are required by the safety standards that are introduced in Sect. 2.2.1. For successfully applying a safety-related system it is important to prove the trustworthiness and acceptability of the solution for the safety-related application problem. Therefore, it must be ensured that the system is stable and meets all given requirements.

In practical application tasks the available training data is often scarce and the number of input dimensions is too large in order to sufficiently apply purely statistical risk estimation methods – for instance K-fold crossvalidation as given in Sect. 2.1.7. Assessing the quality of a learned solution based on the error rate estimated on

a testing set becomes intractable especially for high-dimensional problems because the required error rates cannot be achieved (cf. Tab. 2.1). In many applications, high-dimensional models are required to solve the given problem. Unfortunately, high-dimensional models are hard to verify (*curse of dimensionality*) – a good illustration of this problem is given in (Hastie et al., 2001, Chap. 2.5)), may tend to overfitting, and the interpolation and extrapolation behavior is often intransparent. An illustrative example of a counterintuitive behavior of a classifier is shown in Fig. 1.1. While this classifier achieves a good performance on the training and testing data – there is a sudden change of the prediction of the model within a region not covered by the given data set. Such undesired behavior becomes even more likely and much more difficult to discover in the high-dimensional case. Thus a model building method is required, which guarantees a well-defined interpolation and extrapolation behavior, that is, the decision bounds of the learned models can exactly be determined and evaluated for every point of the input space.

For assessing a data-driven generated solution that is used in safety-related domains it is important to take the following issues into account (Taylor, 2005, Chap. 3):

- The limited size of the testing data set may not allow a proper system evaluation.
- The limited size of the training data set may lead to an inappropriate approximation of the desired function.
- How will the solution deal with previously unseen data?
- The training and testing data might be insufficient – especially for rigorous testing and reliability estimation.
- The training and testing data must represent the entire input domain (completeness and correctness).

Reliability assessment and robustness analysis require a huge number of test cases which may not be available. Insufficient testing data make it impossible to prove the correctness of the learned solution. Furthermore, it must be ensured that proper model assumptions and parameters are chosen.

For successfully applying machine learning within the field of safety-related applications and providing solutions that are accepted by the domain experts the following requirements arise:

Reliability. The learned solution must show a low probability of failure. It must be ensured that the learned solution is always within the specified conditions.

Correct Interpolation and Extrapolation. An unintended and possibly undesired behavior of the learned model must be avoided and a correct generalization behavior must be guaranteed. On the other hand, the solution must be sensitive enough to capture the specific problem.

Data Efficiency. The learning algorithm must be capable of dealing with small training sets, that is, a small number of independent training samples.

Expert Knowledge. It must be possible to include expert knowledge within the model building process.

Interpretability and Verification. Interpretability is essential to facilitate the domain experts to evaluate the decisions of the learned solution. Furthermore, the model should be understandable for people being no experts in the field of machine learning theory.

It is a challenging task to prove that the learned solution is a correct solution of the application problem. The incorporation of a-priori knowledge into the model selection and model learning process can be helpful – for instance one can exploit monotonicity constraints, allow user interactions, or use active learning in order to achieve an improved generalization behavior (Cohn et al., 1994). Nevertheless, the quality of a data-driven generated solution depends on the quality and on the amount of data that is available for training. Unfortunately, within real-world application problems, the number of data available for training is often scarce. For example, a typical data set of the airbag deployment application which is introduced in Sect. 1.2 consists of only 30 to 50 independent crash tests. Therefore, the only way to ensure that the learned solution is a correct solution of the problem is to allow domain experts to evaluate the learned solution according to their domain knowledge.

2.2.3 Machine Learning Approaches for Safety-Related Applications

Lisboa (2001) investigates the current usage of machine learning methods - especially artificial neural networks - in the field of safety-related applications. Recent approaches to adopt machine learning methods for use in safety-related applications can be found, for instance, in Kurd & Kelly (2007); Kurd et al. (2006); Zakrzewski (2001). Furthermore, there are different approaches of providing a verification and validation framework for machine learning methods, for example, Pullum et al. (2007) concern the verification and validation of artificial neural networks.

The important issue of the successful application of machine learning methods in a safety-related domain is to guarantee that the learned solution does not show any unintended inter- or extrapolation behavior – as already mentioned in Sect. 2.2.2 and illustrated in Fig. 1.1. Using visualization techniques developed to map high-dimensional data to a low-dimensional space – for instance multidimensional scaling (Hastie et al., 2001; Rehm et al., 2006), principal curves (Hastie & Stuetzle, 1989), or self-organizing maps (Kohonen, 1990) – might give a good insight into the problem, but there is a loss of information and there is no guarantee that the mapping does not induce or hide inconsistencies of the originally learned solution.

One possibility of providing an interpretable solution is to use rule systems or tree-like models or to extract such representations from non-symbolic models. There are numerous machine learning approaches that can be utilized for doing this –

see, for instance, Andrews et al. (1995); Boz (2002); Breiman et al. (1984); Chen & Wang (2003); Cohen (1995); Kolman & Margaliot (2005); Tzeng & Ma (2005). Taylor (2005) recommends to use rule extraction for generating rule bases from artificial neural networks in order to perform a formal safety analysis of the learned solution. The safety life cycle proposed by Kurd et al. (2006) combines rule extraction and knowledge insertion to an iterative process. It deals with three levels: a symbolic level, a translation level and a neural learning level. The symbolic level is associated with symbolic (i.e., domain) knowledge. The translation level performs the rule insertion and rule extraction to the neural learning level. The neural learning level is used to modify and refine the symbolic knowledge based on the given data. Fuzzy-rule systems (Nauck, 2003; Nauck et al., 1997) are also a good option. They are often applied to controlling tasks. Unfortunately, data-driven methods of building such rule systems often result in huge rule bases which are hard to interpret.

Schlang et al. (1999) combine a radial basis function (RBF) network (Hartman et al., 1990) with an analytical model of the controlled process. The RBF network multiplicatively corrects the output of the analytical model. For unknown inputs it is designed to produce a correction factor close to one so that the output in that case is determined by the analytical model. The advantage of their approach is that the analytical model guarantees a baseline performance which the RBF network can optimize in its trusted input regions. The disadvantage of this approach is that it is only applicable on problems where an analytical model can be provided.

Another possibility to determine a data-driven model is proposed by Zakrzewski (2001). In this approach, an already validated reference implementation (for instance a look-up table) is used as deterministic data generator for an exhaustive evaluation of the machine learning solution. This approach compensates for the limited number of data by generating data according to the reference implementation. Thus it becomes possible to prove that the machine learning solution conforms to the reference implementation. This might be useful to reduce the memory usage by replacing large look-up tables by smaller artificial neural networks but it is questionable why a second model should be built if there exists an already validated reference solution of an application problem.

Confidence measures – for instance error bars provided by Gaussian Processes (GPs) (MacKay, 1998; Rasmussen & Williams, 2006) or multi-layer perceptrons (MLPs) with evidence propagation (MacKay, 1992) – can be used to estimate the uncertainty of the model's output. But for high-dimensional problems where the data might be sparse such confidence measures suffer from the limited number of data that is available. For instance, in the field of aviation and controlling nuclear power plants an error rate of at most 10^{-9} failures per hour is required. Achieving such an error rate by purely assessing the solution based on the error rate estimated on a testing data set and computing some confidence measures becomes intractable for high-dimensional problems.

Another approach of providing an interpretable solution is to use ensemble methods, where the original high-dimensional problem is partitioned into smaller subproblems. Then it becomes possible to visualize and, thus, to interpret the solutions of the smaller subproblems. The methods described in Chap. 3 and Chap. 4 are based on this ensemble modeling idea.

2.3 Summary

This chapter has introduced the basic concepts of machine learning theory and has provided a brief overview of safety-related systems. First, the fundamentals of machine learning theory are given and several common used machine learning methods are briefly introduced, namely basic linear regression, binary classification, support vector classifiers, common extensions of binary classifiers for multi-class problems, and classification and regression trees. These methods will serve as testbed for our methods which are introduced within the following chapters. In the second part of this chapter, a brief overview about safety-related systems is given. Recent safety-standards are briefly reviewed and the problem of assessing solutions for use in safety-related domains is addressed. Different current state-of-the-art methods in machine learning within the domain of safety-related applications are discussed and their advantages and disadvantages are pointed out.

3 Ensembles of Submodels for Safety-Related Classification Problems

In this chapter, we present our classification framework that is designed for use in application domains with very high safety requirements. First, we introduce the basic concepts of our classification framework based on an ensemble of low-dimensional submodels. This approach assumes a binary classification problem. Later within this chapter, we discuss different strategies of extending our ensemble method to deal also with multi-class problems. The ensembles of binary classifiers were developed with the objective of providing interpretable submodels for use in safety-related application domains. The ensembles assume highly imbalanced misclassification costs between the two classes. The extension to multi-class problems is not straightforward because common multi-class extensions might induce inconsistent decisions. We propose an approach that avoids such inconsistencies by introducing a hierarchy of misclassification costs. We will show that by following such a hierarchy it becomes feasible to extend the binary ensemble and to maintain the same desirable properties of the binary ensemble approach.

3.1 Introduction

In Nusser et al. (2007) we proposed a binary ensemble framework for use in safety-related domains. The main design criterion of this approach is to provide an ensemble of binary classification models that use small subspaces of the complete input space enabling the visual interpretation of the models. Since machine learning approaches are regarded with suspicion in the field of safety-related domains the possibility to visualize each submodel greatly facilitates the domain experts' acceptance of the data-driven generated models. An interpretable solution is often required for applications where the available training data is too sparse and the number of input dimensions is too large to sufficiently apply statistical risk estimation methods in practical application tasks. In most cases high-dimensional models are required to solve a given problem. Unfortunately, such high-dimensional models are hard to verify and their interpolation and extrapolation behavior is often unclear. Recall Fig. 1.1 where the prediction of the model changes in a region not covered by the given data set. Such behavior becomes even more likely and much more difficult to discover in the high-dimensional case. Our ensemble approach provides an insight

into each submodel, which can be evaluated according to the given domain knowledge and, thus, the correct interpolation and extrapolation behavior of the model can be guaranteed. The extension of our binary classification ensemble to multi-class problems is not straightforward since commonly used methods like one-against-one or one-against-rest voting (Friedman, 1996; Hsu & Lin, 2002) may introduce inconsistencies – as illustrated in Sect. 2.1.5. We will show that such inconsistencies can be avoided by introducing a hierarchy of misclassification costs. The crucial aspect is to find a suitable trade-off between the generation of an interpretable and verifiable model and the realization of a high predictive accuracy. In most situations, more complex models will be able to achieve a better predictive performance on the available data compared to simpler models. However, a higher complexity of the model will usually lead to an increased effort for model verification.

This chapter is organized as follows. In Sect. 3.2 we recall our binary ensemble approaches for use in safety-related domains. Sect. 3.3 extends the binary classification framework to also solve multi-class problems. Experiments performed on real-world application problems are described in Sect. 3.4. The results of further experiments on well-known benchmark problems are given in Appendix A. Sect. 3.5 summarizes this chapter.

3.2 The Binary Ensemble Framework

This section introduces the basic concepts of our ensemble approach. The algorithms are designed to solve binary classification problems. That is, the learning task is to find an estimate of the unknown function $f : V^N \rightarrow Y$, where $V^N = \times_{n=1}^{N} X_n$ with $X_n \subseteq \mathbb{R}$ is the input space and Y is the target value, given an observed data set $\mathfrak{D} = \{(\vec{v}_1, y_1), ..., (\vec{v}_M, y_M)\} \subset V^N \times Y$.

Basic Idea. Our ensemble framework is originally motivated by Generalized Additive Models (Hastie & Tibshirani, 1990; Stone, 1985) and Separate-&-Conquer approaches (Fürnkranz, 1999). The framework can be interpreted as a variant of the projection pursuit (Friedman & Tukey, 1974; Huber, 1985). Within our framework, we have developed two different ensemble models, the DecisionTree-like Ensemble Model in Sect. 3.2.1 and the Non-hierarchical Ensemble Model in Sect. 3.2.2. Both approaches are designed to find an estimate of the unknown function $f : V^N \rightarrow \mathbb{K}$, where $\mathbb{K} \subset \mathbb{N}$ is the set of class labels. Our approaches are based on the projections of the high-dimensional data to low-dimensional subspaces. *Submodels* g_j are trained on these subspaces. By regarding only low-dimensional subspaces a visual interpretation becomes feasible. Thus the avoidance of unintended extrapolation behavior is possible. The ensemble of submodels boosts the overall predictive accuracy and overcomes the limited predictive performance of each single submodel, while the global model remains interpretable.

Projections of High-Dimensional Data Sets. The projection π maps the N-dimensional input space V^N to an arbitrary subspace of V^N. This mapping is determined by a given index set $\beta \subset \{1, ..., N\}$. The index set defines the dimensions of V^N that will be included in the subspace V_β. Thus the projection π on the input space V^N given the index set β is defined as:

$$\pi_\beta \left(V^N \right) = V_\beta = \times_{n \in \beta} X_n \,. \tag{3.1}$$

Submodels. The j-th submodel is defined as:

$$g_j : \pi_{\beta_j} \left(V^N \right) \rightarrow \mathbb{K}, \tag{3.2}$$

where β_j denotes the index set of the subspace where the classification error of the submodel g_j is minimal. The best projections are determined by a wrapper method for feature selection (Kohavi & John, 1997). This method is discussed in more detail in Sect. 5.1.3. The final function estimate $\hat{f}$ of the global model is determined by the aggregation of the results of all submodels $g_j(\pi_{\beta_j}(\vec{v}))$.

3.2.1 DecisionTree-like Ensemble Model

This approach replaces the binary one-dimensional splits within the nodes of a common classification tree, cf. Sect. 2.1.6, by strong classifiers. In our implementation, we are using support vector machines (SVMs, cf. Sect. 2.1.4), but other classification methods are also viable. The classifiers used as nodes within the tree are restricted to two input dimensions. This facilitates the visualization of the relevant decision region and avoids overfitting. Comparable to the classification tree learner, the best submodel g_j is used to divide the training set into new subsets $\mathfrak{D}_\theta^{new} := \{(\vec{v}, y) | g_j(\pi_{\beta_j}(\vec{v})) = \theta\}$, where $\theta \in \mathbb{K}$. The submodels for these subsets are built recursively until an appropriate termination criterion is fulfilled. The algorithm to build a DecisionTree-like Ensemble Model is given in Algorithm 3.1

There are two possibilities of applying a DecisionTree-like Ensemble Model. The first variant follows the standard classification tree approach and uses the leaf nodes of the tree-like model to predict the final class label. The second variant, which is used in Algorithm 3.2 aggregates all of the votes of each node that are included in the path of the tree when evaluating a novel data point $\vec{v}$. The class label of a novel data point $\vec{v}$ can be determined by aggregating the votes of the nodes for the given sample:

$$\hat{f}(\vec{v}) = \Phi(\{g_j(\pi_{\beta_j}(\vec{v})) | (g_j, \beta_j) \in models\}) \,, \tag{3.3}$$

where Φ is an appropriate aggregation function and $models$ is the set of nodes of tree T being involved by applying Algorithm 3.2 on $\vec{v}$. In our experiments with SVMs as classifiers the following aggregation function provides adequate results:

$$\Phi(\vec{v}) = \frac{1}{|models|} \sum_{i \in models} \frac{1}{1 + e^{2m_i(\pi_{\beta_i}(\vec{v}))}} \,, \tag{3.4}$$

Algorithm 3.1 Building a DecisionTree-like Ensemble Model.

input: data set $\mathfrak{D}$, dim_{limit} – limit of dimensions (fixed)

output: model tree T

function T := buildTree($\mathfrak{D}$)

1: solve $\forall(\vec{v}, y) \in \mathfrak{D} : \min\left\{|y - g(\pi_\beta(\vec{v}))|\right\}$, where $\beta \subset \{1, ..., N\}$ and $|\beta| = dim_{limit}$
2: create new node T
3: $\mathfrak{D}_0^{new} := \{(\vec{v}, y)|g(\pi_\beta(\vec{v})) = 0\}$
4: $\mathfrak{D}_1^{new} := \{(\vec{v}, y)|g(\pi_\beta(\vec{v})) = 1\}$
5: **if** $(\mathfrak{D}_0^{new} \neq \varnothing) \wedge (\mathfrak{D}_1^{new} \neq \varnothing)$ **then**
6: T.model := g
7: T.beta := β
8: T.child[0] := buildTree($\mathfrak{D}_0^{new}$)
9: T.child[1] := buildTree($\mathfrak{D}_1^{new}$)
10: **end if**

Algorithm 3.2 Classifying new samples with a DecisionTree-like Ensemble Model.

input: $\vec{v} \in V^N$ – new sample data point; T – tree returned by Algorithm 3.1

output: class – class prediction of $\vec{v}$

function class := evaluateTree($\vec{v}$, T)

1: votes := evaluateTreeRek($\vec{v}$, T)
2: class := Φ(votes)

function votes := evaluateTreeRek($\vec{v}$, T)

1: g := T.model
2: β := T.beta
3: votes := $g(\pi_\beta(\vec{v}))$
4: **if** $(g(\pi_\beta(\vec{v})) = 0) \wedge (T.child[0] \neq \varnothing)$ **then**
5: votes := [votes evaluateTreeRek($\vec{v}$, T.child[0])]
6: **else if** $(g(\pi_\beta(\vec{v})) = 1) \wedge (T.child[1] \neq \varnothing)$ **then**
7: votes := [votes evaluateTreeRek($\vec{v}$, T.child[1])]
8: **end if**

where $m_i(\pi_{\beta_i}(\vec{v}))$ is the margin of the i-th SVM model on subspace V_{β_i}, represented by $exp_i \in models$ – that is we compute a weighted average of the votes of the experts. Using Eq. 3.4 as aggregation function yields the classification function: $\hat{f}(\vec{v}) = 1 \Leftrightarrow \Phi(\vec{v}) \geqslant 0.5$, otherwise $\hat{f}(\vec{v}) = 0$.

3.2.2 Non-hierarchical Ensemble Model

This ensemble method incorporates prior knowledge about the subgroups of the given problem and avoids hierarchical dependencies of the submodels, which must be taken into account for validating a DecisionTree-like Ensemble Model. It is required that the so-called *default class* c_{pref}, that is, the default state of the safety-

Algorithm 3.3 Building a Non-hierarchical Ensemble Model.

input: data set $\mathfrak{D}$, c_{pref} - label of default class, dim_{limit} – limit of dimensions (fixed)

output: $models$ – set of submodels

function $models$:= buildModel($\mathfrak{D}$, c_{pref})

1: solve $\forall(\vec{v}, y) \in \mathfrak{D} : \min\left\{|y - g(\pi_\beta(\vec{v}))|\right\}$, $\beta \subset \{1, ..., N\}$ s.t. $|\beta| = dim_{limit}$ and $\forall y = c_{pref}$: $|y - g(\pi_\beta(\vec{v}))| = 0$
2: $\mathfrak{D}_{new} := \{(\vec{v}, y) | g(\pi_\beta(\vec{v})) = c_{pref}\}$
3: **if** $(\mathfrak{D} \setminus \mathfrak{D}_{new} \neq \varnothing)$ **then**
4: $\quad models := \{g(\pi_\beta(\cdot))\} \cup$ buildModel($\mathfrak{D}_{new}$, c_{pref})
5: **else**
6: $\quad models := \varnothing$
7: **end if**

Algorithm 3.4 Classifying new samples with a Non-hierarchical Ensemble Model.

input: $\vec{v} \in V^N$ – new sample data point; $models$ – set of models returned by Algorithm 3.3

output: class – class prediction of $\vec{v}$

function class := evaluate_model($\vec{v}$, $models$)

1: class := $\bigvee_{g_j \in models} g_j(\pi_{\beta_j}(\vec{v}))$

related system, must not be misclassified by any of the learned submodels: $\forall y = c_{pref} : \left|y - g\left(\pi_\beta\left(\vec{v}\right)\right)\right| = 0$. This requirement typically leads to imbalanced misclassification costs. The submodels are trained on low-dimensional projections of the high-dimensional input space with the objective to avoid the misclassification of the default class. The submodels greedily separate the samples of the other class from the default class samples. Missed samples of the other class are used to build further sub-experts. The algorithms for building such a Non-hierarchical Ensemble Model and for evaluating a new sample $\vec{v}$ are shown in Algorithm 3.3 and Algorithm 3.4, respectively.

The final function estimate is the disjunctive combination of all learned submodels:

$$\hat{f}(\vec{v}) = \bigvee_{g_j \in models} g_j\left(\pi_{\beta_j}\left(\vec{v}\right)\right), \tag{3.5}$$

where $models$ is the set of all submodels which are returned by Algorithm 3.3. For the sake of simplicity it is defined that the default class c_{pref} is always encoded as 0 and the other class is always encoded as 1 by the submodels g_j.

3.2.3 An Illustrative Example

The CUBES data set is generated from four Gaussian components in a three-dimensional space. This data set is illustrated in Fig. 3.1. For each CLASS 1 cluster 50 sam-

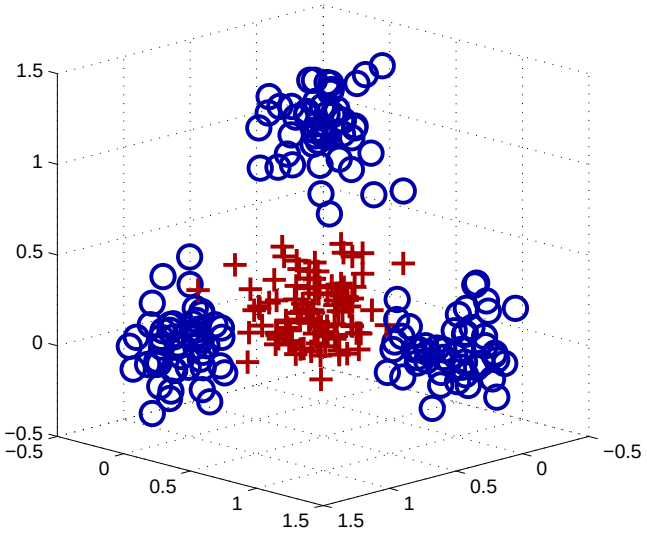
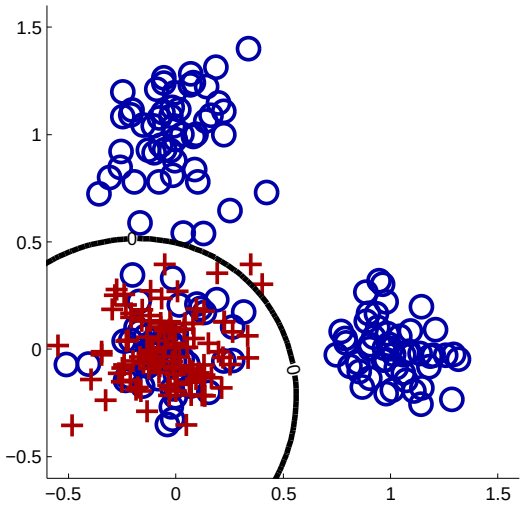


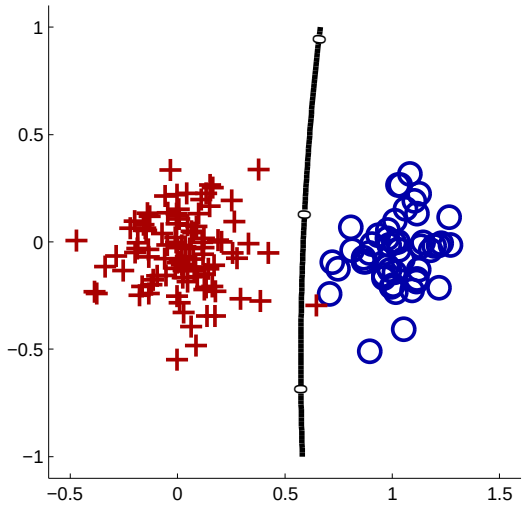
Fig. 3.1: Three-dimensional CUBES data set: CLASS 1 samples are marked with circles and CLASS 0 samples are marked with crosses.

ples are drawn from $\mathsf{N}(e_n, 0.2 \cdot \mathbf{I})$, where e_n is a unit vector and $\mathbf{I}$ is the identity matrix. 100 samples of the CLASS 0 cluster are scattered around the origin, drawn from $\mathsf{N}((0,0,0)^T, 0.2 \cdot \mathbf{I})$. All submodels are trained as SVMs with Gaussian kernels and the parameter set $\gamma = 0.2$ and $C = 5$.

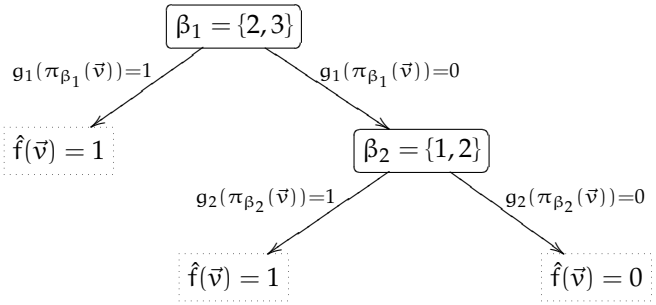
DecisionTree-like Ensemble Model. This method does not require a predefined default class. However, if a default class is given by the application, it can be considered by different misclassification costs when learning the submodels of the tree-like model. In this toy example we ignore the information about the default class. At the initial state, all two-dimensional projections of the CUBES data set are very similar. The best two-dimensional submodel g_1 is depicted in Fig. 3.2(a). It uses the projection $\pi_{\beta_1}(\vec{v})$ with $\beta_1 = \{2,3\}$. This submodel assigns 103 data points to CLASS 1 (2 errors) and 147 data points to CLASS 0 (49 errors). It is not possible to build further submodels for the data points that are assigned to CLASS 1, but for CLASS 0 a second submodel can be built. This model, g_2 with $\beta_2 = \{1,2\}$, is depicted in Fig. 3.2(b). It assigns 50 data points to CLASS 1 (1 error) and 97 data points to CLASS 0 (0 errors). Further improvements are not possible. The final model, which is depicted in Fig. 3.2(c), misclassifies three CLASS 0 samples. All other data points are correctly assigned to their corresponding class labels. The confusion matrix of the final DecisionTree-like Ensemble Model is shown in Tab. 3.1(a).



(a) 1st submodel g_1 with index set $\beta_1 = \{2,3\}$.



(b) 2nd submodel g_2 with index set $\beta_2 = \{1,2\}$.

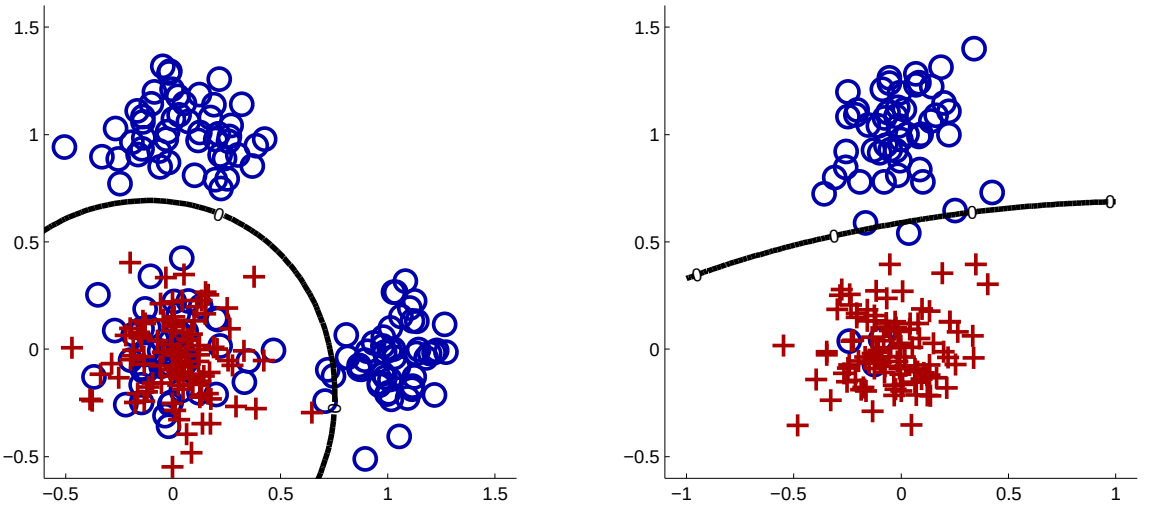


(c) Tree structure of the DecisionTree-like Ensemble Model.

Fig. 3.2: DecisionTree-like Ensemble Model and the CUBES data set: CLASS 1 samples are marked with circles and CLASS 0 samples are marked with crosses. The decision boundaries are drawn as solid lines.

Non-hierarchical Ensemble Model. In this setting, CLASS 0 is chosen as the default class, i.e., $c_{pref} = 0$. So, CLASS 0 must not be misclassified by any learned submodel[1]. This can be achieved, for instance, by using imbalanced misclassification costs for CLASS 1 and CLASS 0. The best submodel g_1, see Fig. 3.3(a), uses the projection $\pi_{\beta_1}(\vec{v})$ with $\beta_1 = \{1,2\}$. 53 data points from CLASS 1 are misclassified by this submodel. Thus in the next iteration new submodels are trained only on samples, which are predicted as CLASS 0 by the first submodel: $\mathfrak{D}_{new} = \{(\vec{v}, y) | g_1(\pi_{\beta_1}(\vec{v})) = 0\}$. In Fig. 3.3(b) the projection $\pi_{\beta_2}(\vec{v})$ with $\beta_2 = \{2,3\}$ of the data set $\mathfrak{D}_{new}$ and the corresponding submodel g_2 are shown. This submodel misclassifies four CLASS 1 samples. Given the chosen parameter set, no further improvements are possible. The final predic-

[1] Note: If one choses CLASS 1 as the default class, $c_{pref} = 1$, it is not possible to solve this learning problem without violating the requirement that the default class must not be misclassified by any submodel. To overcome this limitation we developed a feature construction method based on a multi-layer perceptron. This feature construction method is discussed in Sect. 5.2.



(a) 1st submodel g_1 with index set $\beta_1 = \{1, 2\}$. (b) 2nd submodel g_2 with index set $\beta_2 = \{2, 3\}$.

Fig. 3.3: Non-hierarchical Ensemble Model and the CUBES data set: CLASS 1 samples are marked with circles and CLASS 0 samples are marked with crosses. The decision boundaries are drawn as solid lines.

Tab. 3.1: Confusion matrices of the CUBES data set.

(a) DecisionTree-like Ensemble Model.

	predicted class	
true class	CLASS 0	CLASS 1
CLASS 0	97	3
CLASS 1	0	150

(b) Non-hierarchical Ensemble Model.

	predicted class	
true class	CLASS 0	CLASS 1
CLASS 0	100	0
CLASS 1	4	146

tive model is $\hat{f}(\vec{v}) = g_1(\pi_{\beta_1}(\vec{v})) \vee g_2(\pi_{\beta_2}(\vec{v}))$. The overall performance of the Non-hierarchical Ensemble Model is shown in Tab. 3.1(b): avoiding the misclassification of the default class $c_{pref} = 0$ leads to four misclassified CLASS 1 samples.

3.3 The Multi-Class Ensemble Framework

Our ensemble framework leads to the following two levels where the binary classification approaches can be extended to solve multi-class problems:

1. The multi-class decision is made on the level of the submodels (Ensemble of Multi-Class Submodels, cf. Sect. 3.3.1).

2. Submodels are binary classifiers, the multi-class classification task is performed by the ensemble. Two variants are possible:
   (a) the Hierarchical Separate-and-Conquer Ensemble (cf. Sect. 3.3.2) and
   (b) the One-versus-Rest Ensemble (cf. Sect. 3.3.3).

Another ensemble learning algorithm for multi-class problems is given in Szepannek & Weihs (2006). This algorithm uses a one-against-one approach in order to extend binary classifiers to solve multi-class problems. In this paper the one-against-one method is called *pairwise coupling*. The important similarity of this approach compared to our framework is that it is also based on a reduction of the dimensionality on the level of the submodels. In contrast to our approach, the number of dimensions of the submodels is not limited – all input dimensions that provide statistically sufficient information are included in the training set to build a single submodel to separate the pair of classes. Our ensemble methods may use several submodels with limited dimensionality to solve the same subproblem while each submodel remains visually interpretable.

For safety-related problems it is important to take into account that the commonly used strategies of extending binary classifiers to multi-class classifiers, which are discussed in Sect. 2.1.5 and illustrated in Fig. 2.4, may lead to regions with inconsistent decisions. In order to avoid an unintended labeling the inconsistent decisions are solved according to a hierarchy of misclassification costs: for a given new data point $\vec{v}$ that class label of all predicted class labels is chosen which has the largest misclassification penalty.

3.3.1 Ensemble of Multi-Class Submodels

Using local multi-class models in a *Non-hierarchical Ensemble Model* requires a hierarchy of misclassification costs. That is, we assume that there exists an *ordering* of the class labels, which allows statements like: *"class* c_1 *samples should never be misclassified, class* c_2 *samples might be misclassified only as class* c_1 *samples, class* c_3 *might be classified as class* c_1 *or* c_2 *samples, ..."*

$$\mathtt{penalty}(c_1) > \mathtt{penalty}(c_2) > \mathtt{penalty}(c_3) > ... \tag{3.6}$$

Such a hierarchy of misclassification costs leads to a confusion matrix as depicted in Tab. 3.2. This issue is closely related to ordinal classification problems (Frank & Hall, 2001). An SVM-based approach for ordinal classification can be found in Cardoso et al. (2005).

Combining several local multi-class models becomes difficult because one can only rely on the prediction of the class c_k, which has the minimal misclassification cost – all other class label predictions might be false positives. Thus it is necessary to include all samples that are not predicted as class c_k in the training for the next submodel. This fact leads directly to the Hierarchical Separate-and-Conquer Ensemble approach, which is described in Sect. 3.3.2.

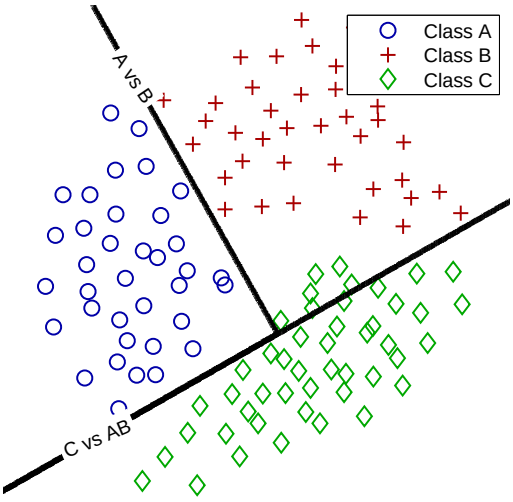
Tab. 3.2: Confusion Matrix for multi-class submodels in a Non-hierarchical Ensemble Model. The following hierarchy of misclassification costs is assumed: $penalty(c_1) > penalty(c_2) > penalty(c_3) > penalty(c_4) > penalty(...)$

	predicted class				
true class	c_1	c_2	c_3	c_4	...
c_1	$h_{1,1}$	0	0	0	...
c_2	$h_{2,1}$	$h_{2,2}$	0	0	...
c_3	$h_{3,1}$	$h_{3,2}$	$h_{3,3}$	0	...
c_4	$h_{4,1}$	$h_{4,2}$	$h_{4,3}$	$h_{4,4}$	...
...	...	...	...	...	...

The extension of the *DecisionTree-like Ensemble Model* in order to solve a multi-class problem is straightforward – a novel subtree is generated for each class predicted by the submodel of the current node. The final classification decision is determined by the leaf node of the learned tree – similar to standard decision tree approaches. In order to avoid inconsistent decisions it is encouraged to also use a hierarchy of misclassification costs in this approach for building the submodels.

3.3.2 Hierarchical Separate-and-Conquer Ensemble

This approach requires a hierarchy of the misclassification costs as already introduced for the Ensemble of Multi-Class Submodels approach. It is related to the commonly used one-against-rest approach. Instead of building all one-against-rest combinations of models, the class with the minimal classification costs is separated from all samples of the other classes via binary submodels. This approach is illustrated in Fig. 3.4. The learning procedure is the same as for the Non-hierarchical Ensemble Model, which is described in Sect. 3.2.2. If the problem is solved for the class with the minimal classification costs or there are no further improvements for this class possible, all remaining samples of this class are removed from the training data set and the procedure is repeated for the class which has now the smallest misclassification costs. This procedure will be repeated until the data set of the next iteration has only a single class label. The resulting binary classifiers are evaluated according to the misclassification hierarchy, that is, in the first step all submodels of the class with minimal misclassification costs are evaluated. If the novel data point cannot be assigned to the class with minimal misclassification costs, the procedure is repeated for the next class within the hierarchy of misclassification costs. If no submodel assigns the novel sample to its class the sample is assigned to the class with maximal misclassification costs.



(a) Discriminant functions.

	predicted class		
true class	A	B	C
A	36	0	0
B	0	38	0
C	0	**6**	41

(b) Confusion matrix.

Fig. 3.4: Hierarchical Separate-and-Conquer Ensemble trained on the data set from Fig. 2.4. The following hierarchy of misclassification costs is assumed: $\mathtt{penalty}(A) > \mathtt{penalty}(B) > \mathtt{penalty}(C)$.

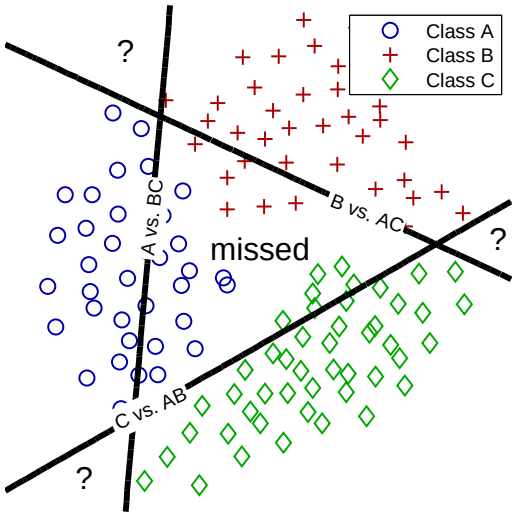
3.3.3 One-versus-Rest Ensemble

This approach follows the one-against-rest multi-class classification approach. It is illustrated in Fig. 3.5. For every class $c_k \in \mathbb{K}$ versus $c_k^* = \mathbb{K} \setminus c_k$ a complete binary Non-hierarchical Ensemble Model $\hat{f}_{c_k}(\vec{v})$ is trained. The class c_k^* is chosen as the default class c_{pref} in order to avoid the misclassification of any sample belonging to $\mathbb{K} \setminus c_k$. For the sake of simplicity c_k^* is encoded as -1. The resulting binary models can be combined by determining the maximum of the predictions of each binary Non-hierarchical Ensemble Model: $\hat{f}(\vec{v}) = \arg\max_{c_k \in \mathbb{K}} \hat{f}_{c_k}(\vec{v})$.

The One-versus-Rest Ensemble is the easiest way to extend our binary ensemble method in order to obtain a multi-class model but it shows a lack of performance for overlapping data sets: it is possible that certain data points will be assigned to the class c_k^* by every submodel and some classes cannot be separated from the other classes due to overlapping of the classes in all projections. This approach still yields ambiguous decisions within the input space, as shown in Fig. 3.5. Such ambiguities can be resolved by the hierarchy of misclassification costs.

3.3.4 An Illustrative Example (Cont'd)

We extend the example which is introduced in Sect. 3.2.3 to a four-class problem: the CLASS 2 samples are drawn from $\mathsf{N}((0.0, 0.0, 0.0)^\mathsf{T}, 0.2 \cdot \mathbf{I})$, the CLASS 3 samples are



(a) Discriminant functions. Ambiguous regions are labeled with '?'.

true class	predicted class A	B	C	?
A	22	0	0	**14**
B	0	31	0	**7**
C	0	0	41	**6**

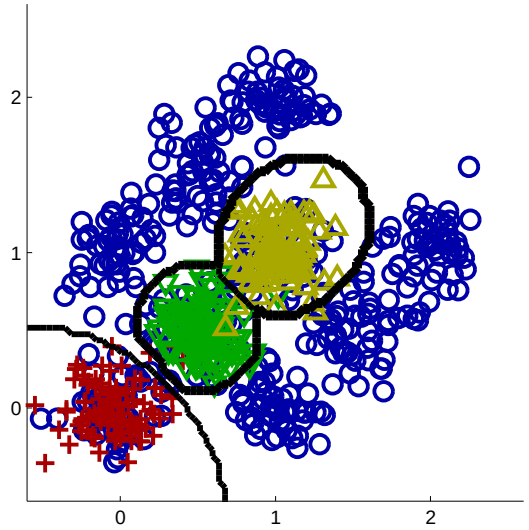
(b) Confusion matrix. The last column denotes missed samples.

Fig. 3.5: One-versus-Rest Ensemble trained on the data set from Fig. 2.4. Each model for class c_k is trained with the objective to avoid the misclassification of all samples belonging to $c_k^* = \mathbb{K} \setminus c_k$.

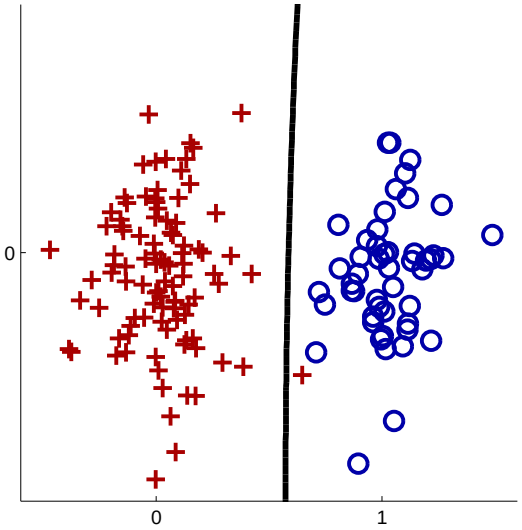
drawn from $\mathsf{N}((0.5, 0.5, 0.5)^\mathsf{T}, 0.2 \cdot \mathbf{I})$, the CLASS 4 samples are drawn from $\mathsf{N}((1.0, 1.0, 1.0)^\mathsf{T}, 0.2 \cdot \mathbf{I})$, and the samples of CLASS 1 are drawn from $\mathsf{N}(e_n + i \cdot 0.5, 0.2 \cdot \mathbf{I}), i = \{0, 1, 2\}$. For this multi-class problem, we assume the following hierarchy of misclassification costs: $\mathit{penalty}(\text{CLASS 4}) > \mathit{penalty}(\text{CLASS 3}) > \mathit{penalty}(\text{CLASS 2}) > \mathit{penalty}(\text{CLASS 1})$.

Ensemble of Multi-Class Submodels. The submodel of the root node of the Ensemble of Multi-Class Submodels approach is shown in Fig. 3.6(a). All predicted CLASS 1 samples are CLASS 1 samples, thus there is no further subtree-building needed for predicting CLASS 1. The second submodel, Fig. 3.6(b), is trained on all samples that are predicted as CLASS 2 by the submodel of the root node. The same holds for the third and fourth submodel, Fig. 3.6(c)&(d), that are trained on the samples predicted as CLASS 3 and CLASS 4, respectively. Further submodels cannot improve the overall performance of the global model. The final classification tree is depicted in Fig. 3.6. It consists of four decision nodes and the maximal tree depth is two, that is, in average two submodels are necessary to classify a novel sample data point.

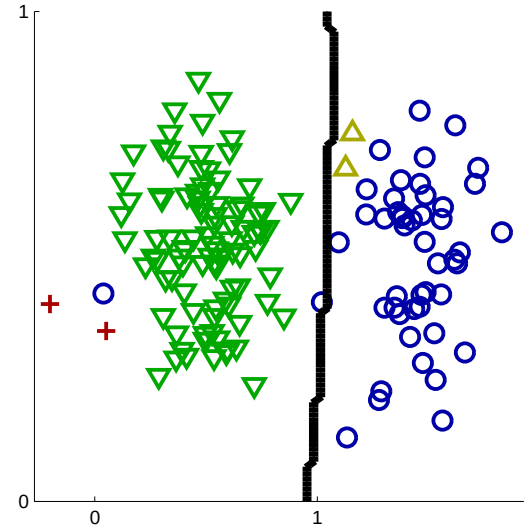
Hierarchical Separate-and-Conquer Ensemble. This approach solves the problem with four submodels, all shown in Fig. 3.8. The first submodel separates most of the CLASS 1 samples from the samples of the other classes. The remaining CLASS 1 sam-



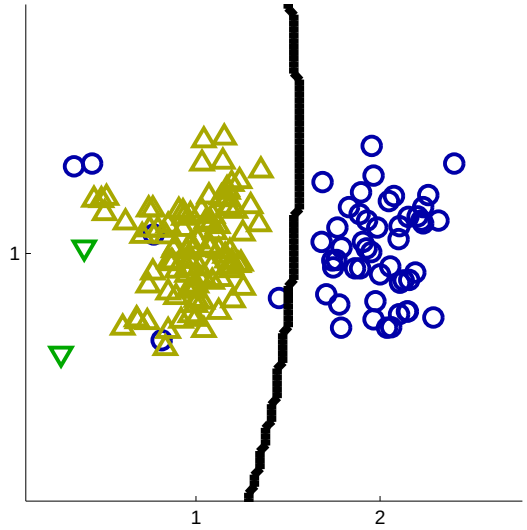
(a) 1st submodel g_1 with index set $\beta_1 = \{2, 3\}$.



(b) 2nd submodel g_2 with index set $\beta_2 = \{1, 2\}$.



(c) 3rd submodel g_3 with index set $\beta_3 = \{1, 2\}$.



(d) 4th submodel g_4 with index set $\beta_4 = \{1, 2\}$.

Fig. 3.6: Ensemble of Multi-Class Submodels approach and the Multi-Class CUBES data set: CLASS 1 samples are shown as circles, CLASS 2 samples are shown as crosses, CLASS 3 samples are shown as downward-pointing triangles, and CLASS 4 samples are shown as upward-pointing triangles. The decision borders of the submodels are drawn as solid lines.

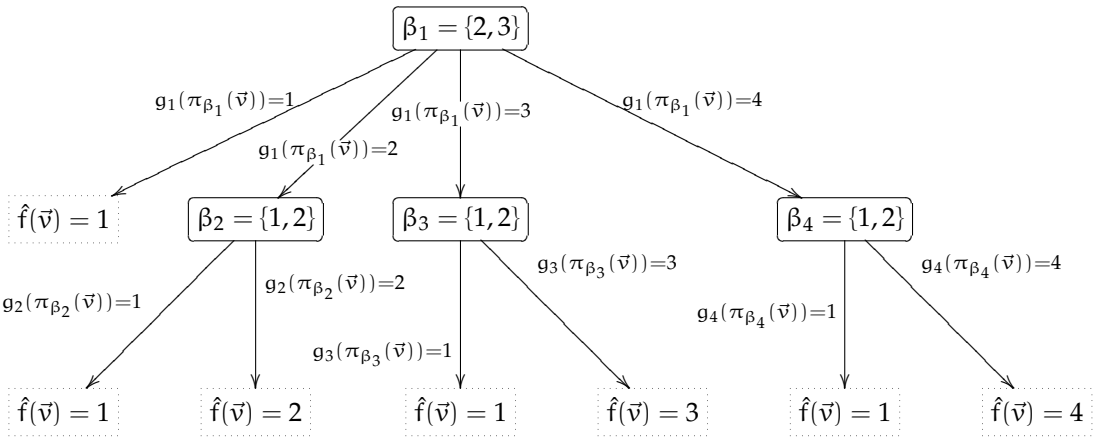


Fig. 3.7: Tree structure of the Ensemble of Multi-Class Submodels trained on the Multi-Class CUBES data set.

ples are removed by the second submodel – further improvements in predicting CLASS 1 are not possible. Thus according to the hierarchy of misclassification costs, the third submodel separates the samples drawn from CLASS 2 from the samples of CLASS 3 and CLASS 4. The last submodel separates the CLASS 3 samples from the CLASS 4 samples.

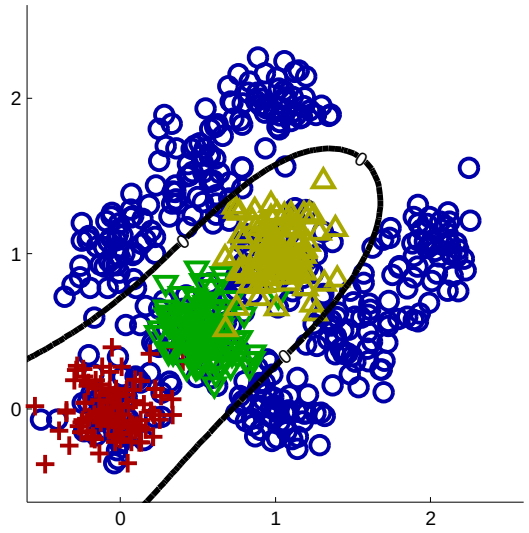
One-versus-Rest Ensemble. This example shows the limitations of the One-versus-Rest Ensemble approach: it is not possible to build one-versus-rest models for CLASS 2, CLASS 3, and CLASS 4 without misclassifying samples from CLASS 1. The only models returned by this approach are the same as shown in Fig. 3.8(a) and Fig. 3.8(b), that is, only CLASS 1 samples can be predicted correctly, all other samples are predicted as 'don't know'.

3.4 Real-World Application Problems

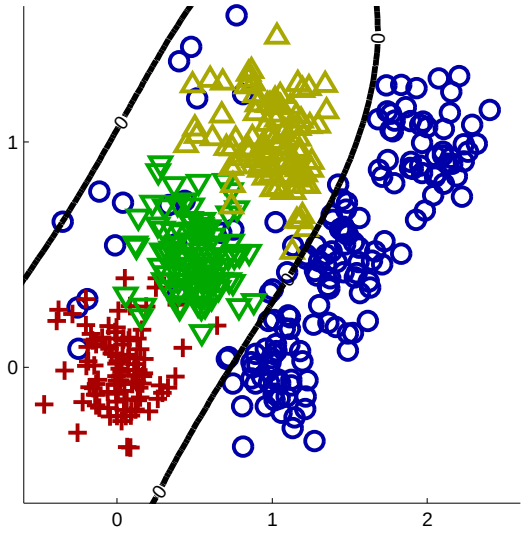
This section discusses the application of our ensemble modeling framework on two real-world application problems with different safety-requirements. The first application serves as an example for application problems with very high safety-requirements. The second application illustrates that the desirable properties of the binary classification ensemble are maintained by the multi-class extensions. Further experiments performed on common benchmark data sets are given in Appendix A.

3.4.1 The Deployment of an Airbag

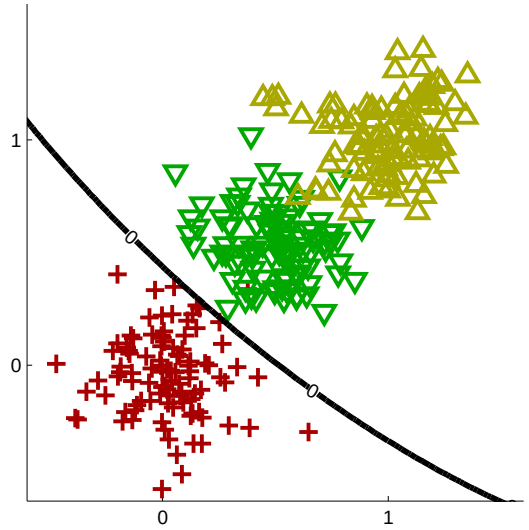
This application problem serves as an example for control systems with very high safety requirements. For each new car platform the control logic of the restraint



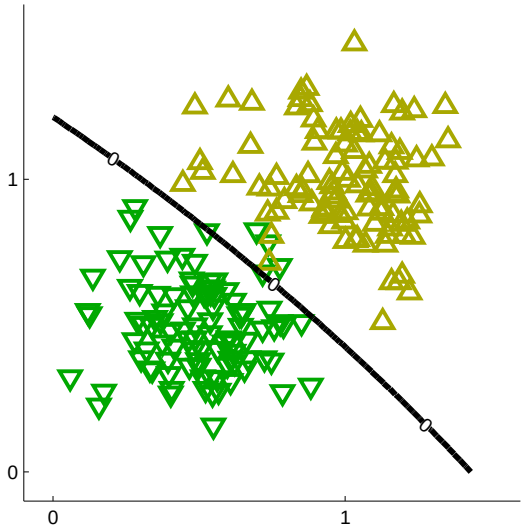
(a) 1st submodel g_1 with index set $\beta_1 = \{2, 3\}$.



(b) 2nd submodel g_2 with index set $\beta_2 = \{1, 3\}$.



(c) 3rd submodel g_3 with index set $\beta_3 = \{1, 2\}$.



(d) 4th submodel g_4 with index set $\beta_4 = \{1, 3\}$.

Fig. 3.8: Hierarchical Separate-and-Conquer Ensemble and the Multi-Class CUBES data set: CLASS 1 samples are shown as circles, CLASS 2 samples are shown as crosses, CLASS 3 samples are shown as downward-pointing triangles, and CLASS 4 samples are shown as upward-pointing triangles.

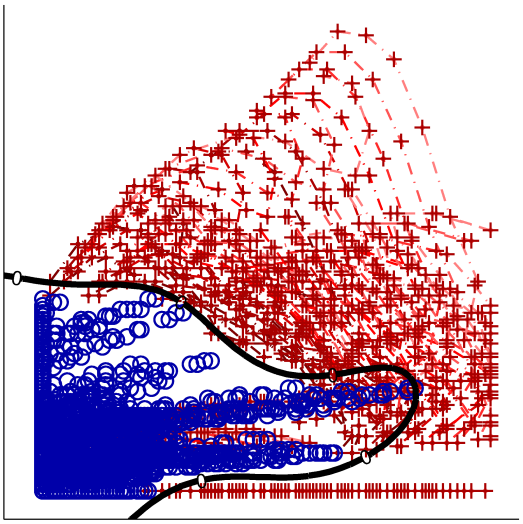
system has to be developed from scratch since modifications of mechanical components, a different sensor placement, or new functional requirements (for instance pedestrian protection) can dramatically influence the signal characteristics and, thus, a solution of the previous platform will not be applicable anymore. Until now most of this calibration is done manually by the safety engineers. An automated process of building such control systems will significantly reduce the development time and costs for a new car platform.

In this problem, based on a high-dimensional data set, a decision has to be derived whether to trigger the restraint system of a car (FIRE) or not (NOFIRE). An example for such a restraint application is the deployment of an airbag system. This classification task is challenging because (1) the restraint system can be triggered only once – a wrong decision cannot be rectified – and (2) a malfunction of the system might be fatal. Thus it must be ensured that the obtained model is sensitive enough to trigger the restraint system and robust enough in order to avoid an unindented extrapolation or interpolation behavior – as illustrated in Fig. 1.1. In order to avoid such an undesired behavior the domain experts in the field of automotive safety electronics tend to use conservative models (for instance rule-based systems or lookup tables). Unfortunately, by generating such models by purely data-driven methods, the resulting models/rule bases become complex and hard to verify.

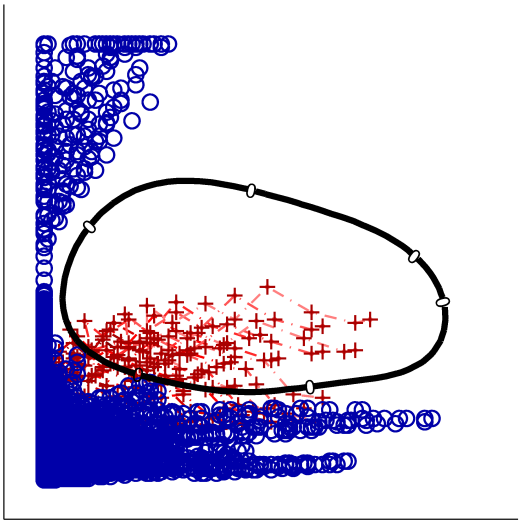
The data set in this application example, which was first used in Nusser et al. (2007), is sparse and consists of approximately 40 000 data points, where each data point is 30-dimensional. These data points belong to 40 distinct time series. Each time series represents a certain standardized crash situation. Due to the limited number of crash tests the guaranteed extrapolation (and interpolation) behavior of the models becomes essential. As the data set consists of a number of time series it is sufficient to trigger the restraint system once in a defined time interval. The incorporation of domain knowledge facilitates the reduction of the model complexity by dividing the FIRE class into two subgroups (FIRE.1 and FIRE.2). These subgroups represent different types of crash situations which implicate input signals with different signal characteristics. Thus in order to reduce the complexity of the learning problem, we can separate the original problem into two distinct subproblems (FIRE.1 versus NOFIRE and FIRE.2 versus NOFIRE) and solve these problems independently. NOFIRE is chosen as the default class. That is, all NOFIRE samples have to be correctly classified by all submodels. NOFIRE is encoded as 0 and FIRE is encoded as 1.

This binary classification problem can be solved by an ensemble of four two-dimensional submodels – two submodels for the FIRE.1 subgroup and two submodels for the FIRE.2 subgroup. The prediction of the global model is simply determined by the disjunction of the predictions of all submodels:

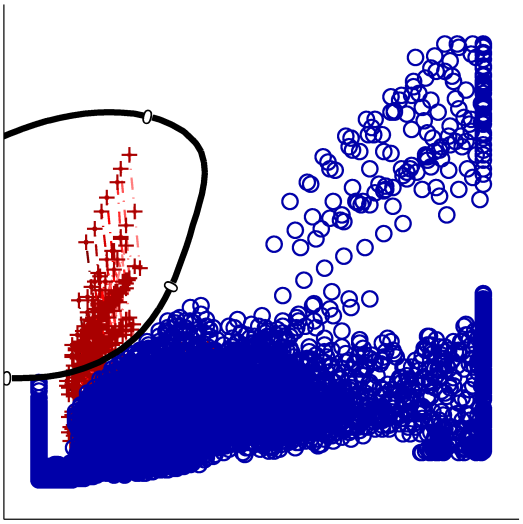
$$\hat{f}(\vec{v}) = \left(g_1^{F1}(\pi_{\beta_1^{F1}}(\vec{v})) \vee g_2^{F1}(\pi_{\beta_2^{F1}}(\vec{v}))\right) \vee \left(g_1^{F2}(\pi_{\beta_1^{F2}}(\vec{v})) \vee g_2^{F2}(\pi_{\beta_2^{F2}}(\vec{v}))\right) .$$



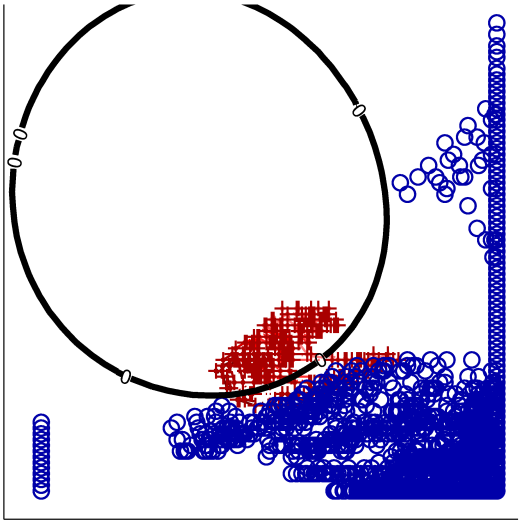
(a) Submodel for FIRE.1-crashes g_1^{F1} with index set $\beta_1^{F1} = \{2, 24\}$.



(b) Submodel for FIRE.1-crashes g_2^{F1} with index set $\beta_2^{F1} = \{2, 27\}$.



(c) Submodel for FIRE.2-crashes g_1^{F2} with index set $\beta_1^{F2} = \{7, 27\}$.



(d) Submodel for FIRE.2-crashes g_2^{F2} with index set $\beta_2^{F2} = \{3, 25\}$.

Fig. 3.9: Non-hierarchical Ensemble Model and the autonomous control example. NOFIRE samples are marked with circles and FIRE samples are marked with crosses. The trajectories of the FIRE samples are shown as broken lines. The decision boundaries of the submodels are drawn as solid lines.

The learned submodels per subgroup are illustrated in Fig. 3.9. All submodels have smooth decision boundaries that are adequate according to domain knowledge. No NoFire sample is triggered. Due to the capability to divide the data set into two different subgroups and to combine the learned submodels disjunctively, all Fire samples can be triggered by at least one expert.

The Non-hierarchical Ensemble Model solves this challenging classification problem and conforms to all given requirements. The visualization of each submodel facilitates the domain experts to perform a direct evaluation of the learned solution. The simple aggregation of the submodels by a disjunctive combination greatly facilitates the interpretation of the global model as well.

3.4.2 A Medical Diagnosis Example

The NewThyroid data set can be obtained from the UCI Machine Learning Repository (Asuncion & Newman, 2007). This application concerns a typical medical data screening problem. The classification task is to predict whether a patient's thyroid belongs to the class euthyroidism (normal = Class 1), hyperthyroidism (hyper = Class 2) or hypothyroidism (hypo = Class 3). The data set consists of 215 instances and each instance is described by five attributes. These attributes are:

T3-resin: T3-resin uptake test (a percentage). The T3 resin uptake test measures the level of thyroid hormone-binding proteins in the blood.

Thyroxin: Total serum thyroxin (T4) measured by the isotopic displacement method.

Triiodothyronine: Total serum triiodothyronine (T3) measured by radioimmuno assay.

Basal TSH: Basal thyroid-stimulating hormone (TSH) measured by radioimmuno assay.

Diff TSH: Maximal absolute difference of TSH value after injection of 200 mg of thyrotropin-releasing hormone compared to the basal value.

The data set is illustrated in Fig. 3.10. The following hierarchy of misclassification costs is assumed: $penalty(\text{HYPO}) > penalty(\text{HYPER}) > penalty(\text{NORMAL})$. That is, Class 3 samples must not be misclassified by any submodel, Class 2 samples might be misclassified as Class 3 samples only, and Class 1 samples might be misclassified as Class 3 or Class 2 samples. The idea behind this hierarchy is to avoid any misclassification of sick patients. Changing the hierarchy of misclassification costs between Class 3 and Class 2 does not influence the results because both classes are well-separated.

Ensemble of Multi-Class Submodels. The Ensemble of Multi-Class Submodels is a straightforward extension of the DecisionTree-like Ensemble Model that uses multi-class submodels within the inner nodes of the tree structure. Like the DecisionTree-like Ensemble Model, this approach does not require a hierarchy of misclassification costs. Therefore, the given hierarchy is ignored while building the Ensemble

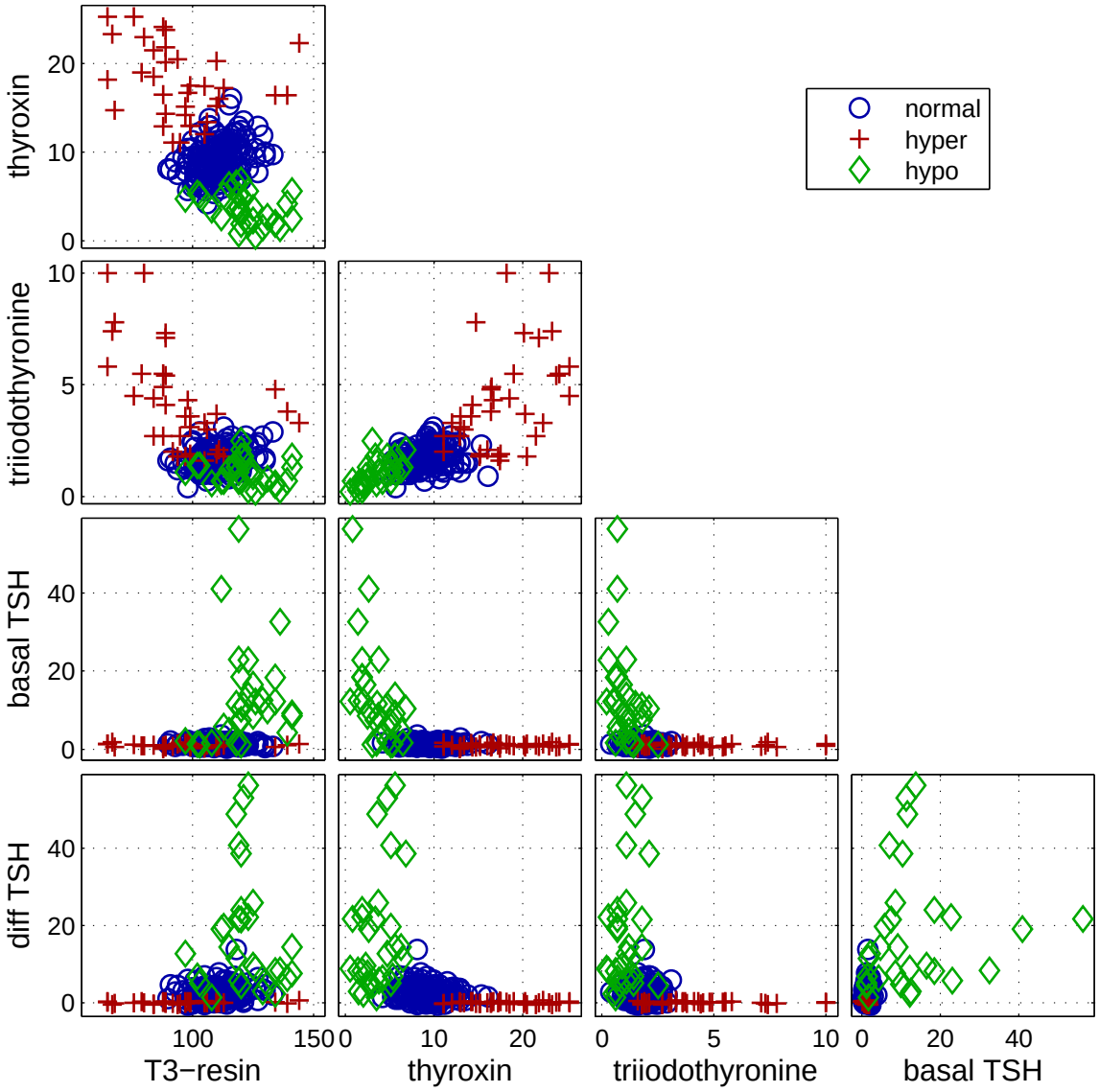


Fig. 3.10: Scatter plot matrix of the five-dimensional NEWTHYROID data set.

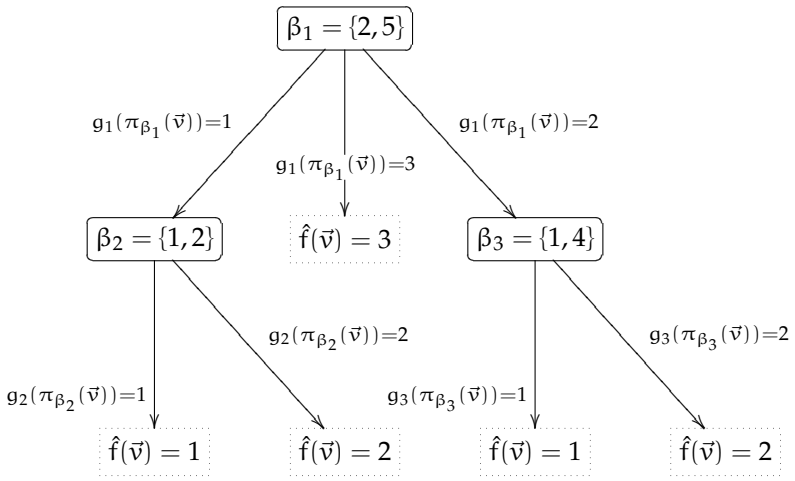
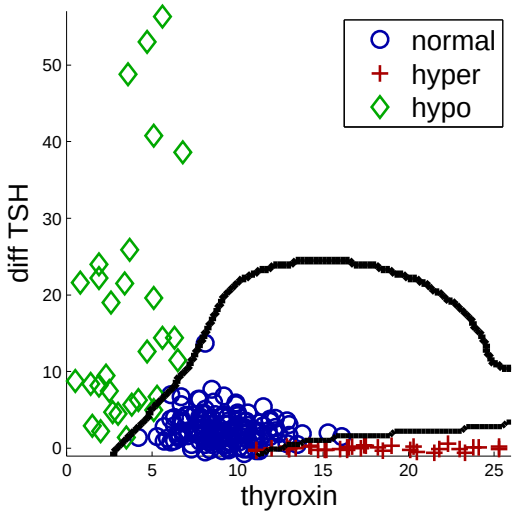


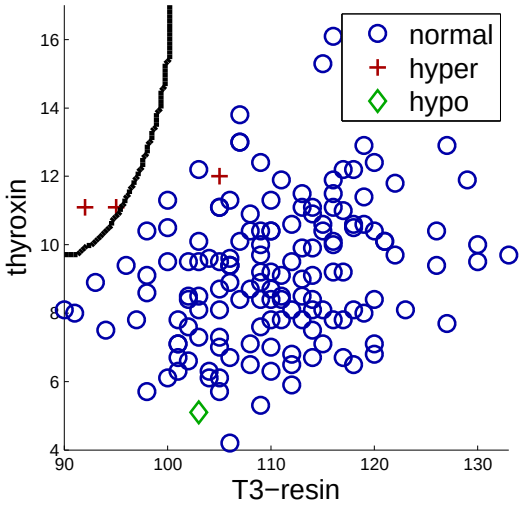
Fig. 3.11: Tree structure of the Ensemble of Multi-Class Submodels trained on the NEWTHYROID data.

of Multi-Class Submodels. The learned solution consists of three submodels. The first submodel is illustrated in Fig. 3.12(a). This model misclassifies only five data points. On the samples that are predicted as CLASS 1 samples a second submodel is learned. This model is depicted in Fig. 3.12(b). On the samples that are assigned to CLASS 2 a third submodel is learned, which is shown in Fig. 3.12(c). All samples that are assigned to CLASS 3 by the first submodel are correctly classified. Thus for the prediction $g_1(\pi_{\beta_1}(\vec{v})) = 3$ no further submodels are required. The final model, which is illustrated in Fig. 3.11, misclassifies only two samples – as one can see in Fig. 3.12(d). Both samples violate the hierarchy of misclassification costs (which was ignored during training). That is, this method still provides a good solution in situations where no further information about the misclassifcation costs are available.

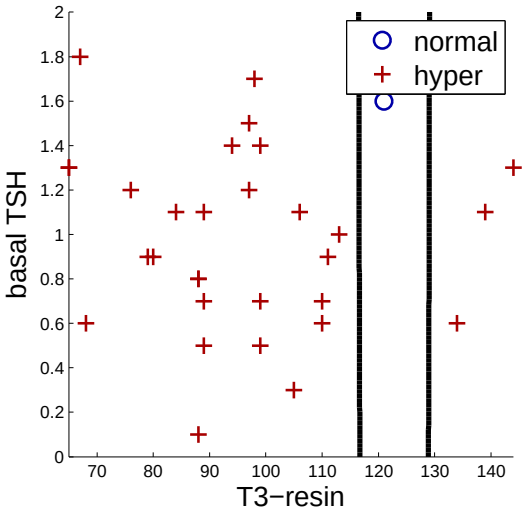
Hierarchical Separate-and-Conquer Ensemble. This approach results in three binary classification submodels which are illustrated in Fig. 3.13. Following the given hierarchy of misclassification costs the first submodel separates most of the CLASS 1 samples from the other ones using the input dimensions "Thyroxin" and "Diff TSH". Further CLASS 1 samples are captured by the second submodel, which uses the input dimensions "T3-resin" and "Thyroxin". The remaining CLASS 1 samples cannot be separated by further two-dimensional submodels. Thus according to the hierarchy of misclassification costs the CLASS 2 samples are separated in the next iteration from the remaining classes (that is, the CLASS 3 samples). The CLASS 2 samples are well-separable from the CLASS 3 samples by setting a simple threshold for the input dimension "Thyroxin" as in the third submodel. As one can see in Fig. 3.13(d), the Hierarchical Separate-and-Conquer Ensemble misclassifies only four instances, which belong all to CLASS 1. No sick patient is predicted as healthy.



(a) 1st submodel g_1 with index set $\beta_1 = \{2, 5\}$.



(b) 2nd submodel g_2 with index set $\beta_2 = \{1, 2\}$.

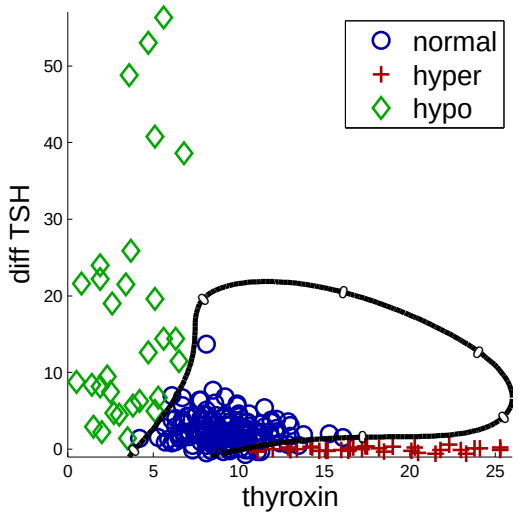


(c) 3rd submodel g_3 with index set $\beta_3 = \{1, 4\}$.

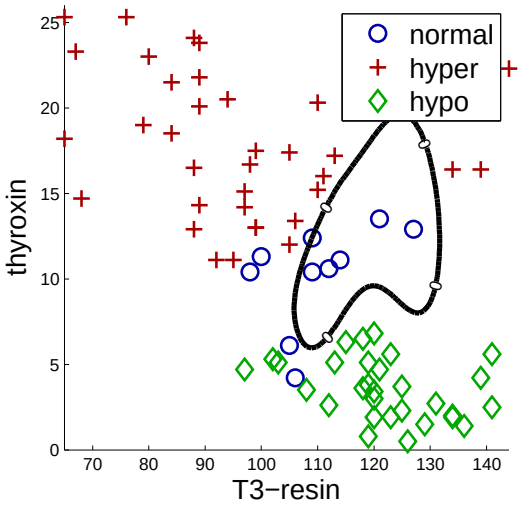
	predicted class		
true class	CLASS 3	CLASS 2	CLASS 1
CLASS 3	29	0	**1**
CLASS 2	0	34	**1**
CLASS 1	0	0	150

(d) Confusion matrix: Ensemble of Multi-Class Submodels and the NEWTHYROID data.

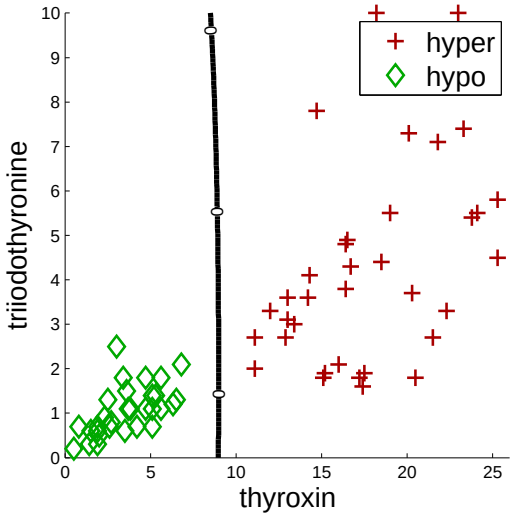
Fig. 3.12: Ensemble of Multi-Class Submodels and the NEWTHYROID data.



(a) 1st submodel g_1 with index set $\beta_1 = \{2, 5\}$.



(b) 2nd submodel g_2 with index set $\beta_2 = \{1, 2\}$.



(c) 3rd submodel g_3 with index set $\beta_3 = \{2, 3\}$.

	predicted class		
true class	CLASS 3	CLASS 2	CLASS 1
CLASS 3	30	0	0
CLASS 2	0	35	0
CLASS 1	**2**	**2**	146

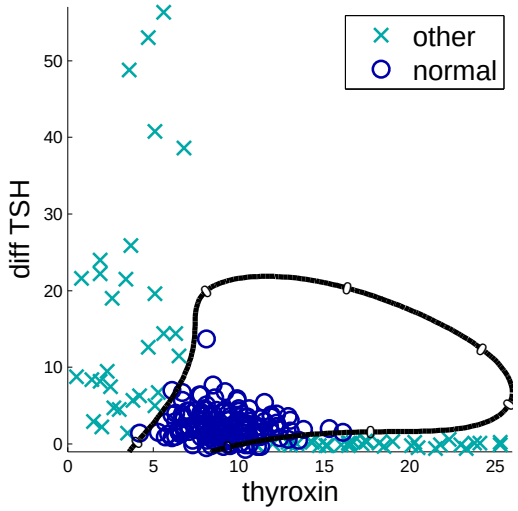
(d) Confusion matrix: Hierarchical Separate-and-Conquer Ensemble and the NEWTHYROID data.

Fig. 3.13: Hierarchical Separate-and-Conquer Ensemble and the NEWTHYROID data.

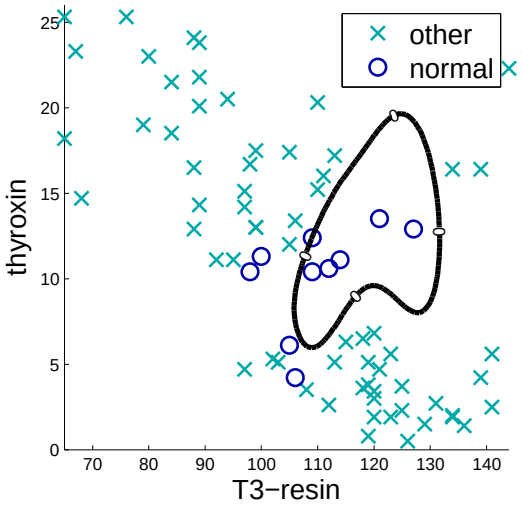
One-versus-Rest Ensemble. This classification approach ends up with four submodels. These submodels are shown in Fig. 3.14. The One-versus-Rest Ensemble follows the one-against-rest strategy, that is, submodels are built for each class in order to separate this class from the samples of all other classes – with the important restriction of never misclassifying the samples that belong to the "other" class. Both the first and the second submodel of this approach are identical with the first and second model of the Hierarchical Separate-and-Conquer Ensemble. The third model of the One-versus-Rest Ensemble separates the CLASS 2 samples from all other samples and the fourth submodel separates the CLASS 3 samples from all samples that do not belong to CLASS 3. As one can see in Fig. 3.14(e), the One-versus-Rest Ensemble misses 10 instances. All other samples are correctly assigned to their corresponding classes. This property of the One-versus-Rest Ensemble is advantageous especially for medical application problems, where further examinations can be carried out in order to judge the patient's status.

Discussion of the Results. Tab. 3.3 summarizes 10-fold-crossvalidation runs that are performed to estimate the error rate of the ensemble methods on previously unseen data. Both ensemble methods are compared with a high-dimensional SVM solution (we used the libSVM implementation of Chang & Lin (2001) with Gaussian kernel) and a classification tree (`treefit` in Matlab). Both standard classification models are trained with imbalanced misclassification costs in order to reproduce the hierarchy of misclassification costs. We computed two performance measures to evaluate the learned models, (1) the predictive error $\widehat{err}$ as defined in Eq. 2.24 on p. 20 and (2) the critical error $\widehat{err}_{crit}$, which counts all samples that violate the given hierarchy of misclassification costs.

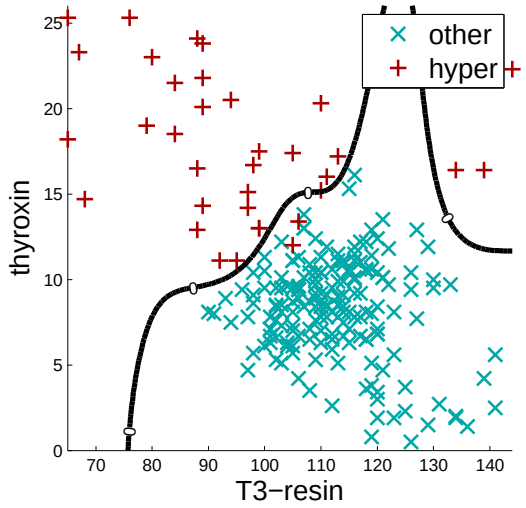
The Hierarchical Separate-and-Conquer Ensemble (HSCE) achieves the best predictive performance on the NEWTHYROID data set. The best critical error rate is achieved by the One-versus-Rest Ensemble (OvRE). The high error rate of the One-versus-Rest Ensemble is due to the fact that missed samples (in average, 4.7% of the data points are missed by the One-versus-Rest Ensemble) are considered as misclassified within the crossvalidation experiments. Nevertheless, for screening medical data the information that an automated classification system cannot assign a sample to a certain class is a benefit as long as the method has a small critical error rate, because in such a situation further examinations can be carried out to ensure the patient's condition. The Ensemble of Multi-Class Submodels (EMCS) shows the worst critical error rate since this approach ignores the given hierarchy of misclassification costs. Compared to the five-dimensional support vector machine the ensemble models are superior with respect to the interpretability of the models because each two-dimensional model can be easily interpreted. The model complexity of the `treefit` model with six decision nodes is comparable to the ensemble models. The disadvantage of the `treefit` model is that the decision nodes depend on their antecessors – while each submodel of the One-versus-Rest Ensemble and Hierarchical Separate-and-Conquer Ensemble can be interpreted independently.



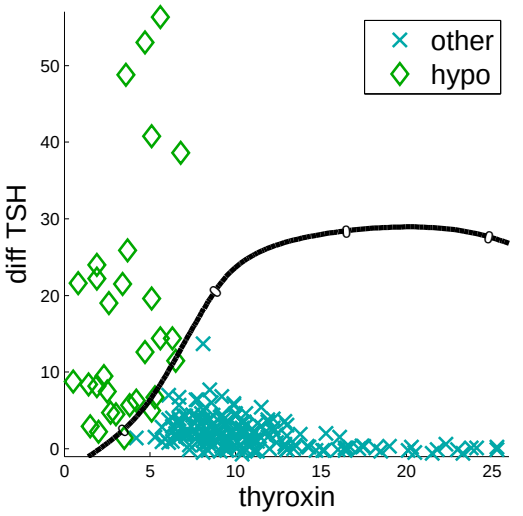
(a) 1st submodel g_1 with index set $\beta_1 = \{2, 5\}$.



(b) 2nd submodel g_2 with index set $\beta_2 = \{1, 2\}$.



(c) 3rd submodel g_3 with index set $\beta_3 = \{1, 2\}$.



(d) 4th submodel g_4 with index set $\beta_4 = \{2, 5\}$.

	predicted class			
true class	CLASS 3	CLASS 2	CLASS 1	missed
CLASS 3	26	0	0	**4**
CLASS 2	0	33	0	**2**
CLASS 1	0	0	146	**4**

(e) Confusion matrix: One-versus-Rest Ensemble and the NEWTHYROID data. The last column shows the samples that are missed by all submodels.

Fig. 3.14: One-versus-Rest Ensemble and the NEWTHYROID data.

Tab. 3.3: NEWTHYROID data set: 10-fold crossvalidation evaluation. #M denotes the number of submodels or decision nodes within the decision tree and #D denotes the dimensionality of each model or decision node, respectively. The $\widehat{err}$ column denotes the overall classification error (including the missed samples of the One-versus-Rest Ensemble). The $\widehat{err}_{crit}$ column denotes the rate of samples that violate the hierarchy of misclassification costs.

Method	#M	#D	$\widehat{err}$ mean (std)	$\widehat{err}_{crit}$ mean (std)
HSCE	3	2	4.19% (5.03)	1.10% (2.86)
EMCS	3	2	4.00% (4.53)	2.57% (3.79)
OvRE	4	2	6.38% (5.63)	0.90% (2.41)
libSVM	1	5	4.62% (4.56)	1.90% (3.71)
`treefit`	6	1	4.29% (4.36)	1.29% (2.12)

3.5 Summary

In order to successfully apply machine learning approaches in the field of safety-related problems it is imperative to provide interpretable and verifiable models. The classification framework proposed within this chapter is based on an ensemble of low-dimensional submodels. Due to limited dimensionality, each submodel can be visualized and hence both be interpreted and validated according to the given domain knowledge. Furthermore, it is possible to avoid unintended and possibly undesired interpolation or extrapolation behavior. The ensemble of the submodels compensates for the limited predictive performance of each single submodel. The proposed ensemble learning methods provide a good trade-off between (1) the interpretation and verification of the learned (sub-) models, avoiding an unintended extrapolation behavior, and (2) the achievement of a high predictive accuracy. In contrast to dimensionality reduction methods, which combine several dimensions of the input space, the submodels are trained on the original dimensions, allowing domain experts to evaluate the trained models directly. By introducing a hierarchy of misclassification costs, which is defined according to the given domain knowledge, it becomes possible to extend our binary classification approach in order to deal also with multi-class problems, while the desirable properties of the binary classification framework can be maintained.

The utilization of our proposed ensemble methods to real-world application problems provide good results. Experiments performed on common benchmark data sets are summarized in Appendix A show that our ensemble models provide a good trade-off between interpretability (measured by the number of submodels and their dimensionality) and the predictive performance of the models.

The described solution of the airbag deployment problem shows the advantages of using our algorithms in an industrial application. The solution is not only available much sooner (compared to the manual calibration performed by the domain experts) but also convinces by its improved performance with respect to crash detection behavior and safety functionality. Despite the sophisticated demands in the sensitive area of passenger safety a verifiably correct solution can be provided by our algorithm.

4 Interpretable Regression Models Based on EM-based Piecewise Linear Regression

This chapter addresses the problem of providing a machine learning method for safety-related regression problems. The problem that arises within the regression setting is that one cannot simply distinguish between a wrong and a correct decision and, thus, it is not possible to force the learning algorithm to always correctly predict a certain state of the system as discussed in Chap. 3[1]. Therefore, the main focus of interest within this chapter is to provide an interpretable solution consisting of simple submodels that can be validated by domain experts and that can achieve an appropriate performance on the application problems. For that reason, it is necessary that the model is simple but at the same time powerful. Piecewise linear regression models can be used to approximate nonlinear functions and these linear submodels show a good interpretability. Unfortunately, the quality of a piecewise linear regression model depends on its partitioning of the input space. However, determining the best partitioning of the input space is a non-trivial problem. This chapter describes an EM-like piecewise linear regression algorithm that uses information about the target variable to determine a meaningful partitioning of the input space. The main goal of this approach is to incorporate information about the target variable in the prototype selection process of a piecewise regression approach. Furthermore, the proposed approach is designed to provide an interpretable solution by restricting the dimensionality of the local regression models. We will show that our approach achieves a similar predictive performance on benchmark problems compared to standard regression methods – while the model complexity of our approach is reduced.

4.1 Introduction

The quality of a piecewise regression algorithm depends on the quality of its partitioning of the input space. Common piecewise regression approaches, for instance

[1] If there is enough information available about the given problem in order to build an analytical model that achieves an appropriate baseline performance one can use commonly used machine learning approaches to enhance the baseline performance within regions where enough data is available. For instance, Schlang et al. (1999) are using RBF networks that can multiplicatively adjust the prediction of the analytical model.

the Local Linear Map (LLM) of Ritter (1991), use only information about the input space for partitioning the data. The target variable is usually ignored while clustering the input space. This strategy becomes inefficient in situations where the data points cannot be distinguished within the input space but where a meaningful partitioning can still be obtained by additionally considering the target variable. Furthermore, regions of high data density in real-world application problems usually correspond to operating points of the system. Such regions are not necessarily appropriate to determine a satisfactory partitioning of the input space because the underlying function of the system might not change within the operating points but in regions with low data density.

The objective of our work within this chapter is to incorporate information about the target variable into the process of choosing the best prototypes of a piecewise linear regression model. In addition, we are interested in providing a suitable trade-off between an interpretable solution and a solution that provides a high predictive accuracy. Symbolic models like regression trees or rule systems allow a good impression about the "big picture" of a given problem. However, they sacrifice some details and usually will not show such a high accuracy as sub-symbolic solutions (for instance neural networks). There are many application domains where finding a good trade-off between interpretability and predictive performance of the learned solution is crucial. For instance, in the field of safety-related applications it is essential to provide transparent solutions that can be validated by domain experts. "Black box" approaches, like artificial neural networks, are regarded with suspicion – even if they show a very high accuracy on the available data – because it is not feasible to prove that they will show a good performance on all possible input combinations. Another example is the field of bioinformatics, where it is necessary to provide transparent solutions to get an impression how the biological mechanisms are working. The problem that arises in both domains is that the amount of independent samples is often not large enough to sufficiently apply statistical risk estimation methods and extensive evaluations. This chapter proposes an EM-like algorithm to determine an interpretable solution based on low-dimensional submodels that achieves a similar performance compared to high-dimensional regression methods. Different approaches were already proposed to tackle the problem of partitioning the data by incorporating the target function: for instance, in (Ferrari-Trecate & Muselli, 2002; Hathaway & Bezdek, 1993) the data is clustered based on the model parameters of local regression models and in Höppner & Klawonn (2003) a combined distance function is proposed for clustering, where the distance within the input space and the error of the submodels is incorporated.

In Sect. 4.2, a brief introduction into the expectation maximization (EM) algorithm, which is a commonly used method of dealing with partially unobservable learning problems, is given. Sect. 4.3 presents our piecewise linear regression approach that is based on the EM algorithm. Furthermore, this section discusses two extensions of this approach: the restriction of the dimensionality of the submodels and the possibility to prune the number of clusters. Experiments performed on two artificial and

on one benchmark data sets are discussed in Sect. 4.4. The results of further experiments performed on other common benchmark problems are given in Appendix A. Sect. 4.6 summarizes this chapter.

4.2 Expectation Maximization

The Expectation Maximization (EM) algorithm is a two-step procedure proposed by Dempster et al. (1977). This approach is capable to deal with learning problems where relevant variables may be unobservable – in our case the correct partitioning of the input space is unknown. The EM algorithm starts with an arbitrary initial hypothesis – for instance a random partitioning of the input space defined by J prototypes. In the first step (the *Expectation* step), the expected values of the hidden (unobservable) variables are estimated under the assumption that the current hypothesis is correct. In the second step (the *Maximization* step), the hypothesis is determined that maximizes the likelihood under the assumption that the hidden variables take the expected values calculated by the first step. Both steps are repeated until the algorithm converges to a local maximum likelihood hypothesis. The example of partitioning the input space based on the EM-algorithm corresponds to the k-means clustering algorithm (MacQueen, 1967).

4.3 The LinEM-Algorithm

The algorithm described in the following can be seen as an adaptation of the EM algorithm. The basic idea of our approach is to incorporate information about the target variable while dividing the input space into different regions, each described by a prototype and a corresponding local regression model. The regression models are trained for regions of the input space. Such regions are represented by cluster prototypes. For each cluster region a linear regression model is learned. Comparable to the EM algorithm, our approach consists of two main steps: first, the local regression models are trained according to the cluster prototypes and, second, the cluster prototypes are updated according to the predictive performance of the linear regression models. The data points are assigned to the linear regression model with the best predictive performance. Then the cluster prototypes are updated to the mean of all samples that are assigned to the corresponding linear regression model.

Learning the Submodels. Given an N-dimensional input space: $V^N = X_1 \times X_2 \times ... \times X_N = \times_{n=1}^{N} X_n$, where $X_\iota \subseteq \mathbb{R}$. The target variable $Y \subseteq \mathbb{R}$ is determined by the (unknown) function: $f : X_1 \times X_2 \times ... \times X_N \rightarrow Y$. The learning task is to determine a function estimate $\hat{f} : V^N \rightarrow Y$ of the (unknown) function f given an observed data set $\mathfrak{D} = \{(\vec{v}_1, y_1), ..., (\vec{v}_M, y_M)\} \subset V^N \times Y$.

The LinEM algorithm is given in Algorithm 4.1. It consists of two main steps: in the first step local regression models are determined according to a given cluster assignment, and in the second step the given data points are assigned to the clusters where the predictive error of the corresponding regression model is minimal. Our algorithm requires a predefined number of clusters $J < M$ that are assumed to represent the data. Each sample data point is assigned to one single cluster. This assignment is defined by the mapping:

$$C(m) = j\,, \quad \text{with} \quad 1 \leqslant m \leqslant M, 1 \leqslant j \leqslant J, \tag{4.1}$$

which assigns the m-th observation to the j-th cluster. Each submodel consists of a tuple of a cluster prototype $\vec{p}_j$ and a local regression model:

$$g_j : V^N \to \mathbb{R}\,. \tag{4.2}$$

The local regression model of the j-th cluster is trained on all data points that are assigned by the mapping C to the j-th cluster:

$$g_j(\vec{v}_m) = y_m\,, \text{where } C(m) = j\,. \tag{4.3}$$

The j-th cluster prototype is determined by:

$$\vec{p}_j = \frac{1}{M_j} \sum_{C(m)=j} \vec{v}_m\,, \tag{4.4}$$

where $M_j = \sum_{m=1}^{M} I(C(m) = j)$ is the number of samples that belong to the j-th cluster and $I(A) = \begin{cases} 1 & \text{if} \quad A \text{ is true} \\ 0 & \text{if} \quad A \text{ is false} \end{cases}$.

The LinEM algorithm performs two alternating updates of the mapping C. Firstly, in line 4 the mapping is updated according to the minimal predictive error of the current local regression models g_j. Then the new prototypes are determined according to Eq. 4.4. Based on the new prototypes and in order to achieve a crisp partitioning of the input space the second update of the mapping C is performed in line 6 of Algorithm 4.1. The data points are assigned to the cluster with the closest cluster prototype:

$$C(m) = \underset{j=1,\ldots,J}{\arg\min}(\mathtt{dist}(\vec{v}_m, \vec{p}_j))\,, \tag{4.5}$$

where $\mathtt{dist}$ is the squared Euclidian distance, $\mathtt{dist}(\vec{v}, \vec{u}) = \|\vec{v} - \vec{u}\|^2$.

The corresponding objective function of the LinEM algorithm is:

$$\mathfrak{J} = \sum_{m=1}^{M} \sum_{j=1}^{J} I(C(m) = j) \left| y_m - g_j(\vec{v}_m) \right| . \tag{4.6}$$

Algorithm 4.1 The LinEM-Algorithm

input: data set $\mathfrak{D}$, number of clusters J

output: cluster prototypes $\vec{p}_j$, submodels g_j

1: Choose random cluster assignment $C(m) = j$
2: **repeat**
3: Learn J submodels according cluster indicies

$$g_j(\vec{v}_m) = y_m\,,$$

where $j = 1, ..., J$ and $C(m) = j$.
4: Update cluster indicies according to model error:

$$C(m) = \underset{j=1,...,J}{\arg\min} \left| y_m - g_j(\vec{v}_m) \right|$$

5: Compute cluster prototypes

$$\vec{p}_j = \frac{1}{M_j} \sum_{C'(m)=j} \vec{v}_m\,,$$

where $M_j = \sum_{m=1}^{M} I(C'(m) = j)$
6: Update cluster indicies according to prototypes

$$C(m) = \underset{j=1,...,J}{\arg\min}(dist(\vec{v}_m, \vec{p}_j))$$

7: **until** termination criterion fulfilled (e.g. maximum of iterations)

Applying the Submodels. In the case of a crisp cluster assignment, the prediction is straightforward:

$$\hat{f}(\vec{v}) = g_{j^*}(\vec{v})\,, \tag{4.7}$$

where $j^* = \underset{j=1,...,J}{\arg\min} \left\{ dist(\vec{v}, \vec{p}_j) \right\}$. If there are reasonable smoothness assumptions about the target function, one can also apply a smoothing function on the predictions of the submodels to determine the global model prediction. As an example, one can use the softmax function:

$$w_{m,j} = \frac{\exp(-dist(\vec{v}_m, \vec{p}_j))}{\sum_{j=1}^{J} \exp(-dist(\vec{v}_m, \vec{p}_j))} \tag{4.8}$$

to compute weights of the submodels. Thus the final model prediction is computed as the weighted sum:

$$\hat{f}(\vec{v}_m) = \sum_{j=1}^{J} w_{m,j} \cdot g_j(\vec{v}_m)\,. \tag{4.9}$$

Restricting the Dimensionality of the Submodels. Adopting the idea used in Chap. 3, instead of using all input dimensions in each submodel, projections of the high-dimensional input space can be used to increase the interpretability of the learned submodels. The projection π maps the N-dimensional input space V^N to an arbitrary

subspace of V^N. This mapping is determined by a given index set $\beta \subset \{1,\ldots,N\}$. The index set defines the dimensions of V^N that will be included in the subspace V_β. The projection π on the input space V^N given the index set β is defined as in Eq. 3.1. Thus the j-th submodel can be re-defined as:

$$g_j : \pi_{\beta_j}\left(V^N\right) \rightarrow \mathbb{R}\,, \tag{4.10}$$

where β_j denotes the index set of the subspace where the predictive error of the submodel g_j is minimal. The best projection can be determined, for instance, by a wrapper model selection method (Kohavi & John, 1997). This ensures that the best possible subspace is used.

Cluster Pruning. The LinEM algorithm facilitates a cluster pruning strategy where clusters can be removed which are ill-posed, that is, the number of data points belonging to the current cluster is too small to solve the regression problem. Furthermore, clusters with similar regression models can be merged. These heuristics enables the LinEM algorithm to deal with problems where the optimal number of clusters is unknown.

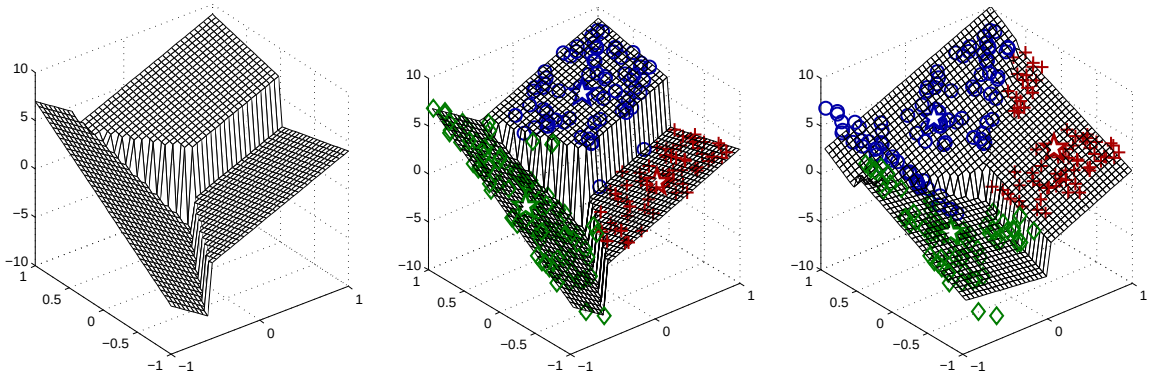
4.4 Two Illustrative Examples

The LinEM algorithm is compared with the Local Linear Map (LLM) from Ritter (1991), a standard linear regression method (`robustfit` in Matlab), and a regression tree (`treefit` in Matlab) on artificial and on common benchmark data sets. We distinguish the crisp and a softmax (c.f. Eq. 4.8) variant of the LinEM algorithm. The predictive error is estimated by: $\widehat{err} = 1/M \sum_{m=1}^{M} |y_m - \hat{f}(\vec{v}_m)|$. The submodels of the LinEM algorithm are estimated by the `robustfit` method and are restricted to two dimensions in all experiments. The best two-dimensional input combination is determined by a wrapper method for feature selection (Kohavi & John, 1997) that performs an exhaustive search through all pairwise combinations. In the following, we discuss two artificial problems in more detail. Further experiments performed on common benchmark data sets are given in Appendix A.

ARTIFICIAL1 Data Set. We used the following function from Ferrari-Trecate & Muselli (2002) as target function:

$$f_1(x_1,x_2) = \begin{cases} 3+4x_1+2x_2 & \text{if} \quad A_1 \wedge A_3 \\ -5-6x_1+6x_2 & \text{if} \quad \neg A_1 \wedge \neg A_2 \\ -2+4x_1-2x_2 & \text{if} \quad A_2 \wedge \neg A_3 \end{cases},$$

where $A_1 : 0.5x_1 + 0.29x_2 \geqslant 0$, $A_2 : 0.5x_1 - 0.29x_2 \geqslant 0$, and $A_3 : x_2 \geqslant 0$. This target function is depicted in Fig. 4.1(a). 300 samples are drawn uniformly from



(a) Target function. (b) LinEM prediction. (c) LLM prediction.

Fig. 4.1: ARTIFICIAL1 data set. The original data points are labeled according to their cluster assignment. The cluster prototypes are depicted as stars. The function estimates of both regression methods are drawn as wireframe surfaces.

$V^N = [-1,1] \times [-1,1]$ and y is determined as $y = f_1(x_1, x_2) + \epsilon$, where $\epsilon \sim N(0, 0.1)$. Note, that in this setting it is impossible to determine appropriate cluster prototypes without regarding the target value.

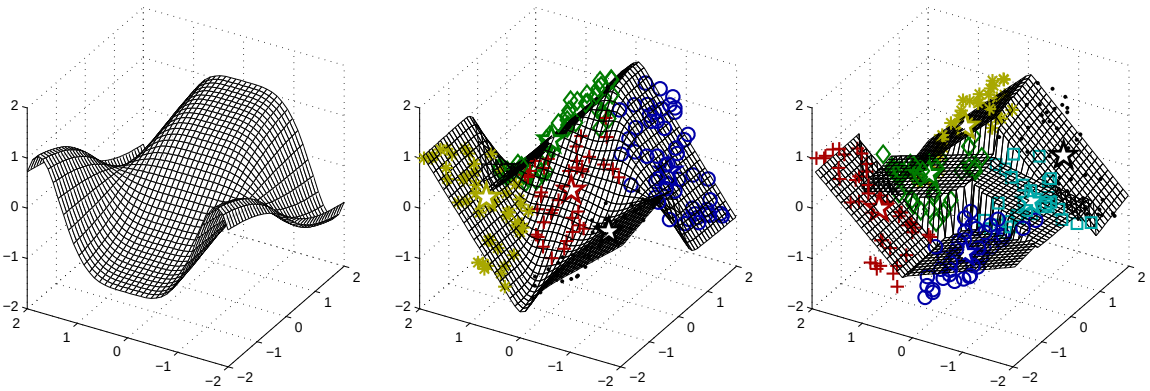
The LinEM algorithm yields the following function estimate:

$$\hat{f}_1(x_1, x_2) = \begin{cases} 3.020 + 3.965x_1 + 1.956x_2 & \text{if} \quad \vec{p}_1 \\ -5.014 - 6.015x_1 + 6.028x_2 & \text{if} \quad \vec{p}_2 \\ -1.999 + 3.998x_1 - 2.028x_2 & \text{if} \quad \vec{p}_3 \end{cases},$$

where $\vec{p}_1 = (0.466, 0.476)^T$, $\vec{p}_2 = (-0.583, 0.122)^T$, and $\vec{p}_3 = (0.426, -0.564)^T$. This function is almost equivalent to the original target function. The function estimate and the corresponding cluster assignments are illustrated in Fig. 4.1(b). Fig. 4.1(c) illustrates the function obtained by applying the LLM algorithm on the same data set. The LLM was unable to determine appropriate cluster prototypes because the prototypes are determined only on the input space. The LLM assumes a wrong partitioning of the input space and, thus, determines inappropriate local regression models.

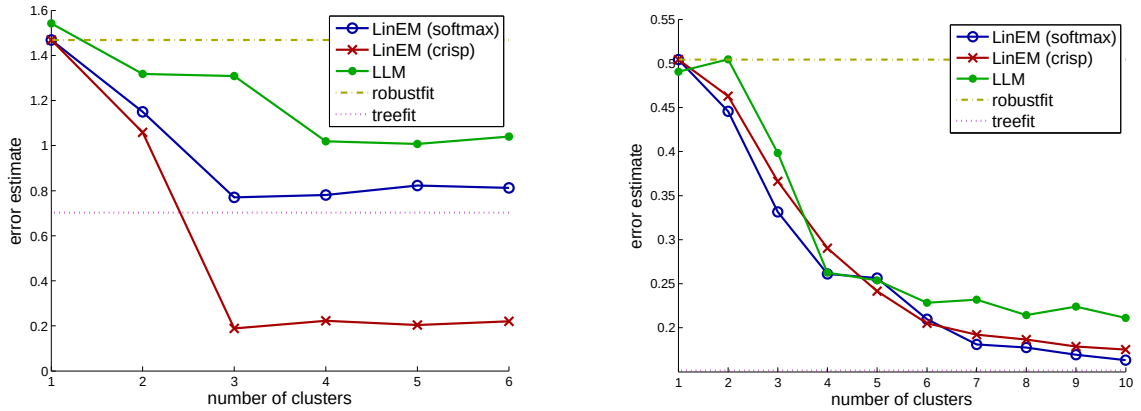
Fig. 4.3(a) illustrates the influence of the number of predefined clusters: with one single cluster the LinEM is equivalent to the `robustfit` solution. The only approach that outperforms the regression tree model (`treefit`) is the crisp LinEM approach. Due to the cluster pruning of the LinEM approach the error estimate remains almost constant for initializations of the number of clusters $k \geqslant 3$ – due to the fact that most of the LinEM models are pruned to three different clusters.

ARTIFICIAL2 Data Set. This data set is used in Höppner & Klawonn (2003). The target function is $f_2(x_1, x_2) = \arctan(x_1)\cos({x_2}^2)$. This function is illustrated in Fig. 4.2(a).



(a) Target function. (b) LinEM prediction. (c) LLM prediction.

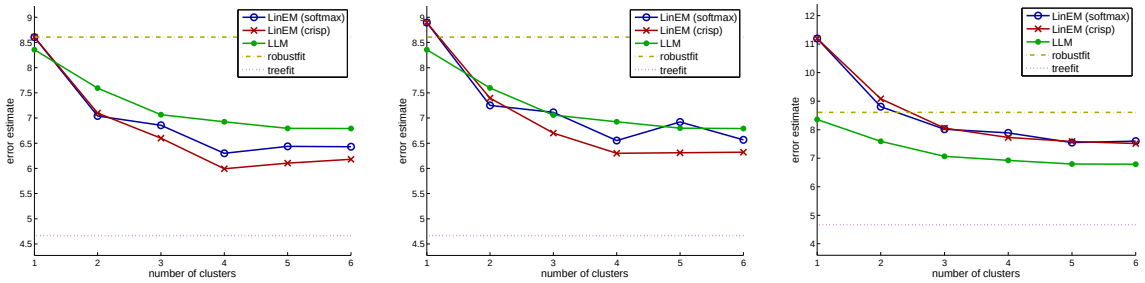
Fig. 4.2: ARTIFICIAL2 data set. The initial number of clusters is set to six for both regression methods.



(a) ARTIFICIAL1 data set. (b) ARTIFICIAL2 data set.

Fig. 4.3: LinEM: Influence of the initial number of clusters on the predictive error. Both data sets are randomly divided into a training data set (80% of the data) and a testing data set (20% of the data). 20 different pairs of training and testing data are generated for each data set. The error is averaged over all testing sets.

300 data points are drawn uniformly from $V^N = [-2,2] \times [-2,2]$ and the value of the target y is determined by $y = f_2(x_1, x_2) + \epsilon$, where $\epsilon \sim N(0, 0.1)$. In Fig. 4.2(b) and Fig. 4.2(c) the model predictions and the corresponding cluster assignments of the data points of the LinEM and LLM solution are shown. Fig. 2.6 depicts a pruned regression tree and its corresponding prediction for this problem. Fig. 4.3(b) illustrates the influence of the initial number of cluster on the predictive error. The performance of the LinEM approach is superior to the performance of the LLM model due to a more intuitive placement of the prototypes. For cluster initialization larger than six the number of clusters is pruned to six clusters.



(a) Unrestricted dimensionality of the LinEM submodels.

(b) The LinEM submodels can use four input dimensions.

(c) The LinEM submodels can use two input dimensions.

Fig. 4.4: Error estimates on CONCRETE COMPRESSIVE STRENGTH data set.

4.5 A Real-World Application Example

The CONCRETE COMPRESSIVE STRENGTH data set was first used in Yeh (1998) and can be obtained from Asuncion & Newman (2007). The task of this data set is to predict the compressive strength of concrete, which is an important and safety-relevant property for civil engineering. It is known that the concrete compressive strength is a highly nonlinear function of the concrete's age and its ingredients. This data set consists of 1030 instances. Each instance is represented by eight attributes, namely cement (kg in a m^3 mixture), blast furnace slag (kg in a m^3 mixture), fly ash (kg in a m^3 mixture), water (kg in a m^3 mixture), superplasticizer (kg in a m^3 mixture), coarse aggregate (kg in a m^3 mixture), fine aggregate (kg in a m^3 mixture), and age (in days). All input dimensions are scaled to the interval $[-1, 1]$ in order to rule out scaling effects. The learning task is to predict the concrete compressive strength measured in mega pascals (MPa).

Within our experiments, the CONCRETE COMPRESSIVE STRENGTH data set is randomly divided into a training data set (80% of the data) and a testing data set (20% of the data). 20 different pairs of training and testing data are generated and for each pair the predictive error is estimated on the testing set. The LinEM algorithm is compared with an LLM, a regression tree (`treefit` in Matlab), and a simple linear regression (`robustfit` in Matlab). For the LinEM, we are using the crisp and the softmax variant. The LinEM is trained with no restriction of the dimensionality of the submodels (that is, with eight-dimensional submodels), with the restriction to four-dimensional submodels, and with the restriction to two-dimensional submodels. The outcome of these experiments is given in Tab. 4.1. The best predictive performance is achieved by the `treefit` method, which yields a complex model with 163 decision nodes. The `robustfit` method yields the worst predictive performance. Both piecewise regression methods (LLM and LinEM) yield intermediate results – these results are not very astonishing, since in is known that the CONCRETE COMPRESSIVE STRENGTH data set consists of a non-linear problem.

Method	#M	#D	$\widehat{err}$ (std) training	testing
LinEM (softmax)	4	8	6.21 (0.26)	6.38 (0.35)
LinEM (softmax)	4	4	6.47 (0.28)	6.55 (0.38)
LinEM (softmax)	5	2	7.53 (0.34)	7.61 (0.50)
LinEM (crisp)	4	8	5.69 (0.16)	6.03 (0.42)
LinEM (crisp)	4	4	6.12 (0.13)	6.30 (0.35)
LinEM (crisp)	5	2	7.40 (0.32)	7.69 (0.71)
LLM	6	8	6.64 (0.23)	6.79 (0.44)
`robustfit`	1	8	8.41 (0.27)	8.60 (0.67)
`treefit`	163	1	*2.30* (0.07)	*4.67* (0.37)

Tab. 4.1: CONCRETE COMPRESSIVE STRENGTH data set and different regression methods. #M denotes the number of submodels or decision nodes within the decision tree and #D denotes the dimensionality of each model or decision node, respectively. The Error column denotes the predictive error estimated on the testing set. The standard deviation of the predictive error is given in parentheses.

The LLM outperforms the LinEM only when the submodels within the LinEM are restricted to two input dimensions. In Fig. 4.4 the influence of the initial number of clusters is illustrated. The cluster pruning yields four clusters for cluster initializations with more than four cluster for the LinEM algorithm – except for the setting where the submodels are restricted to two-dimensional subspaces which leads to five clusters.

The LinEM with a crisp partitioning and with four four-dimensional submodels achieves a good trade-off between predictive performance and the interpretability of the learned solution. This solution outperforms the LLM which uses six eight-dimensional submodels. That is, our algorithm obtains a more appropriate partitioning of the input space, since it is capable to include information about the target variable in the model building process. Furthermore, the restriction of the dimensionality of the submodels and the cluster pruning strategy yield a good interpretability.

4.6 Summary

Providing a regression method for safety-related problems is difficult since one cannot distinguish between a correct and a wrong decision as in Chap. 3. Therefore, it becomes more important to provide an interpretable solution that can be validated by the domain experts. Piecewise linear regression models are known to provide well-interpretable solutions. But the predictive performance of those methods de-

pends on the partitioning of the input space. Common piecewise linear regression methods determine the partitioning of the input space while ignoring the target variable. This method might not be appropriate for a multitude of real-world application problems because regions of a high data density correspond to the working points of a system. Often, such regions are not useful to determine a meaningful partitioning of the input space.

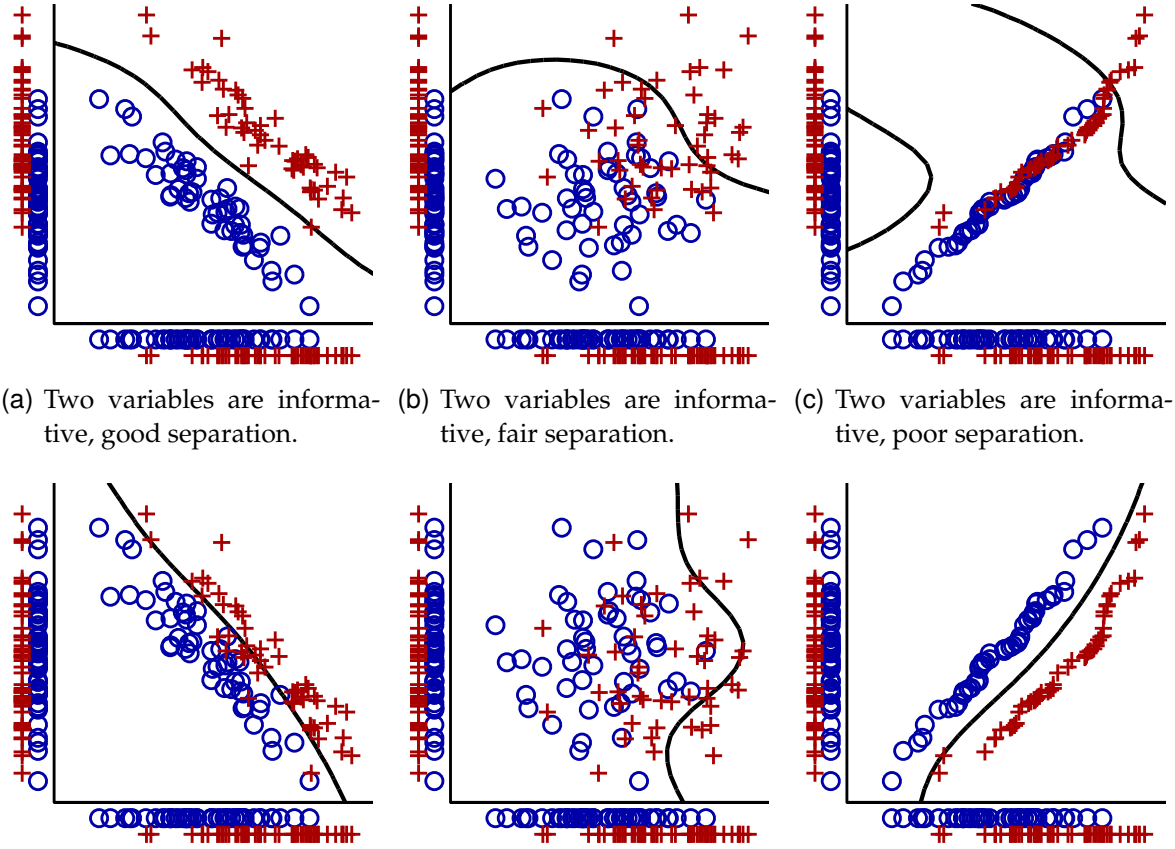
Incorporating information about the target variable into the prototype selection process of a piecewise regression can greatly improve the performance on problems where the input space does not provide sufficient information for partitioning the data. The proposed LinEM algorithm provides a good trade-off between the interpretability and the predictive performance of a learned solution. It achieves a similar performance compared to established regression methods while the model complexity can be reduced. It is capable of reconstructing piecewise linear functions from a given data set. Furthermore, the approach facilitates a cluster pruning strategy which has been proven useful in situations where no apriori information about the number of clusters is available.

5 Feature Extraction and Data Filtering

This chapter discusses different methods that can be used to enhance the predictive performance of the approaches given in Chap. 3 and Chap. 4. The first important aspect concerns feature extraction, that is, the problem of determining appropriate subsets and useful transformations of the input variables, in order to obtain a good solution of the learning problem. Feature extraction can be subdivided into two different task: feature selection and feature construction. A good overview about feature extraction in general is given in Guyon et al. (2006). Feature selection (cf. Sect. 5.1) is an important issue in order to determine the best low-dimensional submodels within our ensemble framework – cf. Chap. 3. Feature construction (cf. Sect. 5.2) is useful in order to overcome possible limitations of the predictive performance that can arise on some application problems due to the limited dimensionality of the submodels. In Sect. 5.3, another approach of simplifying a classification problem is described which is based on filtering the training data set in order to remove conflicts, which can dramatically decrease the predictive performance especially for low-dimensional submodels. Finally, based on a real-world application problem we discuss the suitability of the different methods.

5.1 Feature Selection

In feature selection, which is also known as variable selection or subset selection, one is interested in reducing the size of the input space by removing irrelevant or misleading random variables. That is, only such a subset of the random variables within the input space V^N should be retained that is useful to solve the learning task. According to Chap. 3, feature selection can be seen as the process of determining the best low-dimensional projection of the original input space. Choosing the best low-dimensional projection $\pi_\beta\left(V^N\right)$, cf. Eq. 3.1, in order to determine the submodel g_j is a challenging task. One possibility of choosing the best projection is to rank the input dimensions according their individual relevance and to use only those dimensions which have a high ranking. Such methods are quite fast and are preferable in situations where the number of input variables becomes very large. This approach can provide good results as depicted in Fig. 5.1(a). Nevertheless, it is not sufficient in all situations. Examples of a poor separation on highly ranked



(a) Two variables are informative, good separation.

(b) Two variables are informative, fair separation.

(c) Two variables are informative, poor separation.

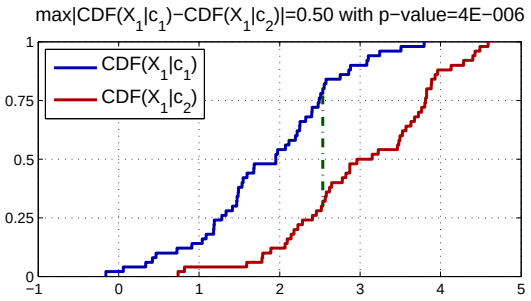
(d) Only one variable is informative, fair separation.

(e) Only one variable is informative, poor separation.

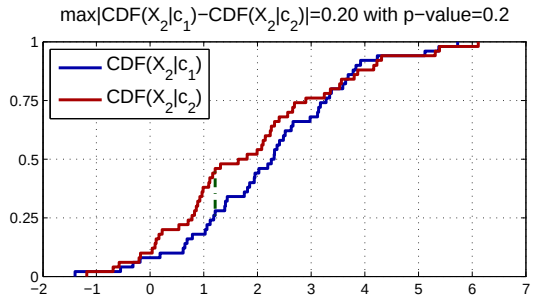
(f) Only one variable is informative, good separation.

Fig. 5.1: Classification problems and feature selection: CLASS 1 samples are marked with circles and CLASS 2 samples are marked with crosses. Projections of each class are drawn beyond the axes. The decision boundary of an SVM classifier with higher misclassification costs for CLASS 1 is drawn as solid line. The projections on the axes are identical for (a),(b),(c) and (d),(e),(f), respectively.

input dimensions are shown in Fig. 5.1(b) and 5.1(c). On the other hand, removing low ranked input dimensions, for instance the second dimension of Fig. 5.1(f), can sacrifice good solutions. It is possible that the best feature combination consists of only very low-ranked input variables. The only way to ensure that the best combination of input variables is determined is to perform a exhaustive search through the space of all possible combinations. That is, feature selection is a challenging problem, which is important in order to obtain a sufficient solution of the learning problem by applying the algorithms of Chap. 3 and Chap. 4.



(a) Kolmogorov-Smirnoff test performed on the 1st input variable.



(b) Kolmogorov-Smirnoff test preformed on the 2nd input variable.

Fig. 5.2: Univariate Kolmogorov-Smirnoff test performed on the data from Fig. 5.1(d),(e),(f) where the projections on the axes are identical. The conditional cumulative probability distributions given the class labels, that is $\mathsf{CDF}(X|c_1)$ and $\mathsf{CDF}(X|c_2)$, are drawn as solid lines. The values of the test statistics are drawn as broken lines.

5.1.1 Feature Selection Based on Univariate Statistical Tests

A simple but quite fast method for feature selection is to perform univariate statistical tests in order to determine variables that are useful to solve the given classification problem. Those methods ignore possible dependencies among the variables, but – especially for a large number of input variables and the assumption that most of these variables are just noise – those methods allow to dramatically reduce the dimensionality of the given problem.

In our experiments we used the two-sample Kolmogorov-Smirnoff test (Feller, 1948; Szepannek & Weihs, 2006) as one example of univariate statistical tests. This test is used in order to compare the marginal distributions given the class labels[1]. The following test statistic is used:

$$D_{KS} = \max |\mathsf{CDF}(X|c_1) - \mathsf{CDF}(X|c_2)| \ , \tag{5.1}$$

where $\mathsf{CDF}(X|c_1)$ and $\mathsf{CDF}(X|c_2)$ denote the conditional empirical cumulative probability distribution of random variable X given class c_1 and c_2, respectively. Examples of those empirical cumulative probability distributions are drawn in Fig. 5.2.

The algorithm for feature selection using the Kolmogorov-Smirnoff test is given in Algorithm 5.1. This algorithm evaluates for each input variable whether there is a significant difference between the conditional cumulative probability distributions. In order to compare both distributions, a hypothesis test is performed with the null

[1] It is also possible to use impurity measures as introduced in Sect. 2.1.6 – for instance Eq. 2.18 or Eq. 2.19. Using those measures is closely related to the feature selection method of Sect. 5.1.2, which can also be used to deal with regression problems.

Algorithm 5.1 Feature selection based on the Kolmogorov-Smirnoff test. (Assumes binary target variable!)

input: data set $\mathfrak{D}$, significance level α for hypothesis testing

output: index set β

function β := KStestFS($\mathfrak{D}$,α)

1: $\beta := \varnothing$
2: **for all** $n \in \{1, \ldots, N\}$ **do**
3: Compute test statistic for variable X_n:

$$D_{KS} := \max |\mathsf{CDF}(X_n|c_1) - \mathsf{CDF}(X_n|c_2)|$$

4: Compute p-value of the observed value of the test statistic (Feller, 1948, Eq. 1.4):

$$p := 1 - 2\sum_{i=1}^{\infty}(-1)^{i-1}\exp\left(-2i^2\left(\sqrt{\frac{M_{c_1}M_{c_2}}{M_{c_1}+M_{c_2}}}D_{KS}\right)^2\right),$$

where M_{c_k} is the number of data points in $\mathfrak{D}$ that belong to class c_k
5: **if** $p < \alpha$ **then**
6: $\beta := \beta \cup \{n\}$ // Null hypothesis H_0 is rejected.
7: **end if**
8: **end for**
9: **if** $\beta = \varnothing$ **then**
10: $\beta := \{1, \ldots, N\}$ // Use all variables if test fails.
11: **end if**

hypothesis H_0 that both conditional distributions are equal. The alternative hypothesis H_A is that both conditional distributions are different. The null hypothesis H_0 can only be rejected if the probability of the observed value of the test statistic D_{KS} becomes unlikely given the distribution of D_{KS} under the assumption of the null hypothesis H_0. The p-value is determined in line 4 according to the distribution of the test statistic D_{KS}. This distribution can be found in the statistical literature – for instance in Feller (1948). The null hypothesis is rejected if the p-value is smaller than the predefined significance level, for instance $\alpha = 0.05$. The input dimension is kept if the null hypothesis can be rejected, otherwise the input dimension is removed from the training set. In the case that all dimensions would be removed from the training set, the algorithm returns all variables in order to allow the subsequent model learning procedure to exploit all possible dimensions.

This procedure is fast but ignoring possible dependencies among the variables may sacrifice a good performance of the subsequent model building procedure. Looking back at Fig. 5.1(d),(e),(f) – the results of the Kolmogorov-Smirnoff test performed on these three problems are identical and are shown in Fig. 5.2. It is possible that variables are removed which might be useful for separating both classes if they are used in conjunction with other features, cf. Fig. 5.1(f). Another problem that might arise when using such univariate tests is that the procedure keeps a large number of vari-

ables, but the possibility of achieving a better performance by allowing more than two or three features sacrifices the visual interpretation of each model, which is the advantage of the ensemble methods described in Chap. 3 and Chap. 4. Nevertheless, this feature selection method can be useful for high-dimensional problems as preceding feature selection. The final feature and model selection should be performed by the method described in Sect. 5.1.3 in order the determine the best possible feature combination.

5.1.2 Feature Selection Based on Classification and Regression Trees

As already pointed out in the previous section, univariate feature selection methods might keep a quite large number of variables that might be redundant or unnecessary in order to solve the given learning problem. The approach discussed within this section is comparable to the previous approach, but it is intended to include only features that are really necessary to explain the structure within the data. Therefore, a classification or regression tree is built (cf. Sect. 2.1.6). The tree learning algorithm greedily determines the best splits of the input space based on a impurity measure. That is, for each node within the tree a greedy feature selection is performed in order to solve a classification or regression problem. The internal feature selection performed by a tree-based model can be exploited in order to select those input dimensions that are necessary to explain the non-random structure of the given data.

This idea of exploiting the internal feature selection of a tree learner in order to perform a feature selection is used, for instance, in Grabczewski & Jankowski (2005). The set of input dimensions that should be retained can be obtained by traversing through all inner nodes of the learned classification and regression tree model. Only the dimensions used within the inner nodes of the tree are kept. The resulting set of variables is usually smaller than the set of variables returned by Algorithm 5.1, but there is still no guarantee that a visualization of the reduced input space becomes possible.

5.1.3 Wrapper for Feature Selection

In order to find the best possible low-dimensional projection of a restricted dimensionality a wrapper-based feature selection method (Kohavi, 1995b; Kohavi & John, 1997) can be used. This method treats the learning algorithm as "black box" – that is, no assumptions about the learning algorithm are required. The wrapper-based feature selection method performs an exhaustive search through all possible feature combinations and uses the learning algorithm itself as part of the evaluation function in order to determine the best subset. That is, for each feature combination within the search space a model is built. The predictive performance of each

Algorithm 5.2 Wrapper based feature selection.

input: data set $\mathfrak{D}$, $\mathtt{dim}_{\mathrm{limit}}$ – number of dimensions

output: index set β, submodel g

function (β, g) := WrapperFS($\mathfrak{D}$,$\mathtt{dim}_{\mathrm{limit}}$)

1: $\beta := \varnothing$
2: Construct list L of all possible combinations of input variables with length $\mathtt{dim}_{\mathrm{limit}}$
3: **for all** $\beta_{\mathrm{tmp}} \in L$ **do**
4: Learn model $g_{\mathrm{tmp}} : \pi_{\beta_{\mathrm{tmp}}}\left(V^N\right) \rightarrow Y$ in order to solve learning problem
5: Evaluate performance of current model g_{tmp}
6: **if** performance of g_{tmp} is better than for all previously tested models **then**
7: $\beta := \beta_{\mathrm{tmp}}$
8: $g := g_{\mathrm{tmp}}$
9: **end if**
10: **end for**

model is determined and the best model and its corresponding feature combination is chosen. Within our experiments it is assumed that the number of features that are included within the feature combinations is fixed – as in Algorithm 3.1 and Algorithm 3.3.

This method has high computational costs but the wrapper-based feature selection is the only variant to guarantee that the best submodel is determined allowing a visual interpretation of the learned solution which is mandatory for successfully applying the algorithms of Chap. 3. To speed up this procedure it is possible to perform a preceding feature selection using the methods described in Sect. 5.1.1 or Sect. 5.1.2 or to use domain knowledge in order to reduce the total number of input dimensions of the learning problem.

5.1.4 Further Feature Selection Methods

Multivariate analyses might be carried out by determining, for instance, the Pearson or Spearman correlation between two variables and in order to remove highly correlated variables from the training set. This is related to the wrapper method described above, because all pairs of variables must be tested. This method also does not guarantee the best solution – for instance in Fig. 5.1(f) both dimensions are highly correlated, nevertheless, using both dimensions for classification one can obtain a perfect separating hyperplane. In Guyon & Elisseeff (2006) you can find a good introduction to feature extraction in general.

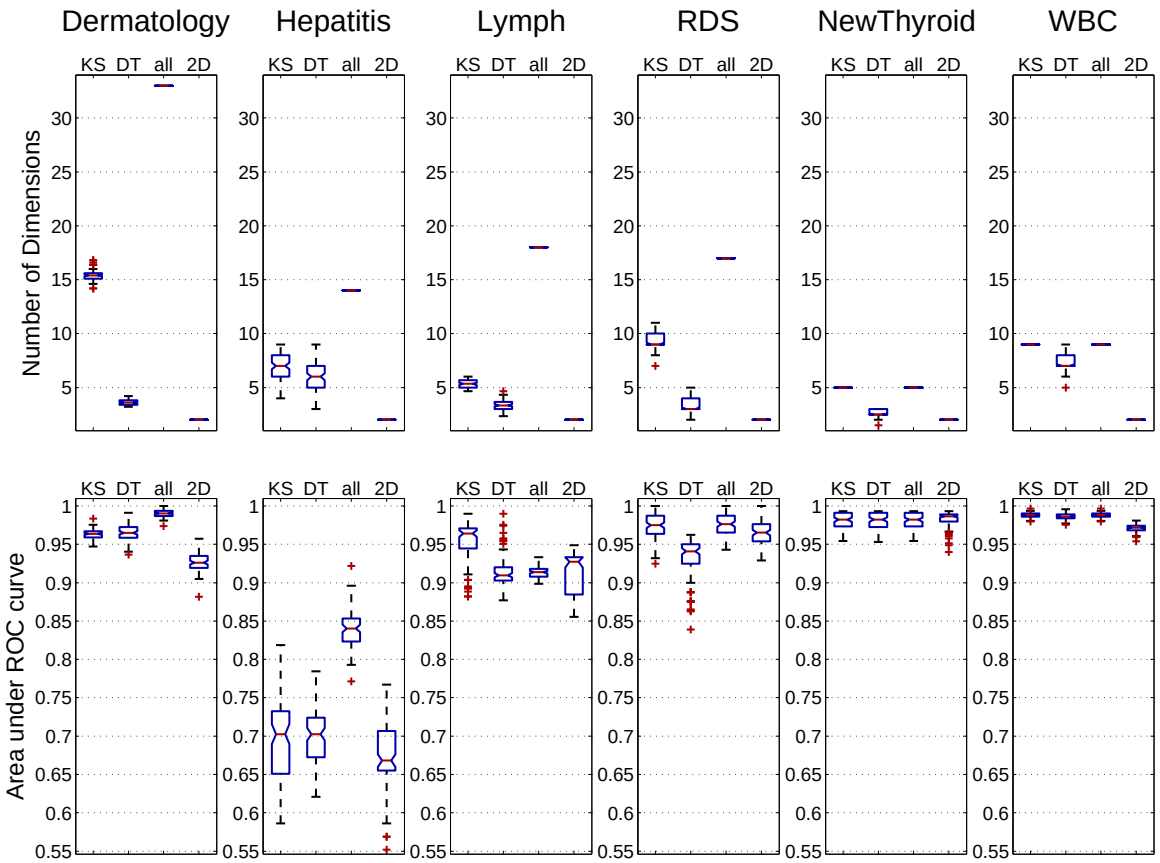


Fig. 5.3: Comparison of different feature selection methods on several benchmark data sets. KS – Kolmogorov-Smirnoff test, DT – decision tree features, all – use always all features, and 2D – use best two-dimensional projection. The models are trained on 75% of the data set and the error is estimated on the remaining 25% of the data set. This procedure is repeated 100 times for each data set and each feature selction method.

5.1.5 Comparison of Different Feature Selection Methods

The influence of the dimensionality of each submodel on the predictive error of the learned ensemble models is investigated by an experimental evaluation carried out on several benchmark data sets. A description of each data set can be found in Appendix A. In this setting, the Hierarchical Separate-and-Conquer Ensemble is extended in order to use further feature selection methods: that is, instead of the wrapper feature selection with the restriction to two-dimensional projections, the algorithm can use the Kolmogorov-Smirnoff test to determine all variables that have different conditional cumulative probability distributions given the class labels (cf. Szepannek & Weihs (2006)) or it can select only those variables that are used within the decision nodes of a CART model (Breiman et al., 1984). That is, the Gini's diversity index serves as split criterion – see Sect. 2.1.6. In this study we divided the data sets into four roughly equally sized folds and repeated the experiments for 100

different fold initializations. As evaluation measure we used the one-point-estimate of the area under the ROC curve (AUC as defined in Eq. B.5), which can be computed for every kind of classifier. The AUC can be interpreted as the probability that a classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance – for details on the analysis of ROC curves, please confer Appendix B. The results of this study are illustrated in Fig. 5.3. All results are obtained by applying the Non-hierarchical Ensemble Model algorithm on the same data sets as in the previous study. For each data set the Non-hierarchical Ensemble Model is trained with four different settings: (1) use Kolmogorov-Smirnoff test to determine all dimensions that have different conditional cumulative distributions per class label, (2) use all dimensions that are used by a decision tree learner, (3) use all dimensions for learning the submodel, and (4) use only two-dimensional projections. Since (1-3) can exploit a higher dimensionality, these variants are restricted to one submodel per class label. There is no general trend which feature selection method performs best in all situations. The best predictive performance depends more on the structure of the data set than on the feature selection method itself. There is also a significant difference between the Kolmogorov-Smirnoff test based feature selection and the decision tree based feature selection method on the LYMPH and RESPIRATORY data sets. If one compares the average dimensionality of the submodels (which can be seen as indicator of the interpretability of the models) for each different feature selection method, the restriction to only two-dimensional submodels is competitive compared to the other approaches and provides an appropriate trade-off between interpretability and predictive performance.

5.2 Feature Construction

In the following we will address two problems that may arise while applying the classification ensembles: (1) due to a large overlap of the classes[2] within all low-dimensional subspaces the submodels might not be capable of finding a good separation of the different classes and (2) the classification problem might not be solved with the restriction to two- or three-dimensional submodels because the underlying function requires a higher dimensionality. The first problem can be tackled by a preceding data filtering. Two examples of such filtering approaches are given in Sect. 5.3. The second problem can be solved by a stacking-like approach (Gama & Brazdil, 2000; Wolpert, 1992), which is discussed in Sect. 5.2: additional input dimensions are generated by a multi-layer perceptron(MLP). These additional input dimensions can be interpreted as preceding soft classifiers.

[2] Such overlap might be induced by noise within the input data or due to an inconsistent labeling of the given data points.

5.2.1 Principal Component Analysis

A widely used method for feature construction is the principal component analysis. This approach generates a set of features that are all linear combinations of all original variables. The basic idea behind this approach is to determine a set of features providing the best possible reconstruction of the original dimensions (Duda et al., 2001, Sect. 3.8). The linear combinations are simple to compute and analytically tractable. Nevertheless, methods that generate linear combinations of all input dimensions are not applicable for safety-related applications because the newly generated features depend on *all* input dimension which violates the requirement of providing an interpretable solution. Therefore, we developed a feature construction method in Nusser et al. (2008b) that is capable of generating low-dimensional linear combinations of the original input dimensions. This method is discussed in the following.

5.2.2 MLP-Based Feature Construction

The Non-hierarchical Ensemble Model and its multi-class extensions show a good performance on real-world applications but there are classification problems which cannot be solved with the restriction to two- or three-dimensional submodels. To overcome this limitation a feature construction method based on a multi-layer perceptron (MLP) architecture[3] has been developed. The additionally generated input dimensions can be interpreted as preceding soft classifiers – a similar approach is proposed by Liu & Setiono (1998), where the hidden units of a fully connected MLP architecture were used to build multivariate decision trees. This method is used as heuristic that performs a fusion of input dimension to overcome the limitations of low-dimensional submodels.

The original input dimensions $V^N = X_1 \times X_2 \times ... \times X_N$ are used in the input layer and the target variable Y is used in the output layer of the MLP. The hidden layer of this network consists of $N(N-1)/2$ nodes, that is $\mathtt{hidden}_{(i,j)}$, where $i, j \in \{1, ..., N\}$ and $i < j$. Each hidden node is only connected to two of the original input dimensions[4]. The connections from the hidden to the output layer are set to 1 and are fixed during the network training procedure. An example of this MLP architecture is depicted in Fig. 5.4.

This idea is similar to the way of extending a linear regression model (cf. Sect. 2.1.2 on p. 12) to a polynomial regression model. Due to the chosen network design the MLP is forced in the hidden layer to find local classifiers on the given input dimen-

[3] Further details about artificial neural networks can be found, for instance, in Nabney (2002). This book provides also a good insight of implementing neural networks in Matlab.

[4] This architecture can be easily extended to linear combinations of more than two dimensions. Nevertheless, we recommend to start with low-dimensional linear combinations and to increase the dimensionality only if it is necessary.

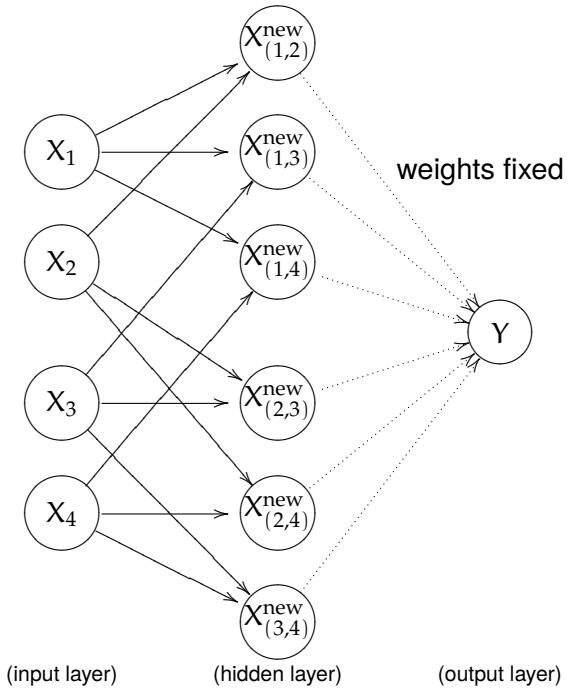


Fig. 5.4: Example of an MLP for feature construction used on a four-dimensional input space.

sion. These classifiers are simple linear models where each uses only two input dimensions. The resulting weights of the hidden neurons are used to build additional input dimensions. An additional input dimension is generated by:

$$X^{new}_{(i,j)} = \tanh\left(X_i \cdot w^{hidden}_{(i,j)} + X_j \cdot w^{hidden}_{(j,i)} + b^{hidden}_{(i,j)}\right), \tag{5.2}$$

where X_i, X_j are the original input dimensions, $w^{hidden}_{(i,j)}$ is the connecting weight of the input dimension X_i to the hidden neuron $hidden_{(i,j)}$, and $b^{hidden}_{(i,j)}$ is the bias of the hidden neuron $hidden_{(i,j)}$. The additional input dimension $X^{new}_{(i,j)}$ can be seen as a preceding soft classifier. Finally, the additional input dimensions are appended to the original data set.

Using all $N(N-1)/2$ additional input dimensions drastically increases the effort to determine the best projections of the data set. Thus it is necessary to reduce the number of additional input dimensions by adding only the "best" hidden neurons as additional input dimensions to the original data set. For instance, such selection can be performed by choosing the hidden neurons which are most correlated with the target variable.

The advantage of the MLP-based feature construction method is that the restricted dimensionality of the linear combinations facilitates the understanding of the additionally generated dimensions. In contrast to the original stacking method (Wolpert,

1992), the additionally generated input dimensions (meta-attributes) are only determined as linear combinations of the original input dimensions. It is not allowed to use the output of a meta-attribute to build an additional meta-attribute. We do not allow a higher complexity of the additionally generated dimensions because an increasing complexity prevents the acceptance of the solution by the domain experts in the field of safety-related problems.

This feature construction method can be modified by building a so-called bottleneck encoder, that is, instead of using the target variable as output of the network the original input dimensions of the data set are used also within the output layer of the MLP. Thus the network is forced to determine low-dimensional linear combinations within the hidden layer that allow the reconstruction of the original input dimensions.

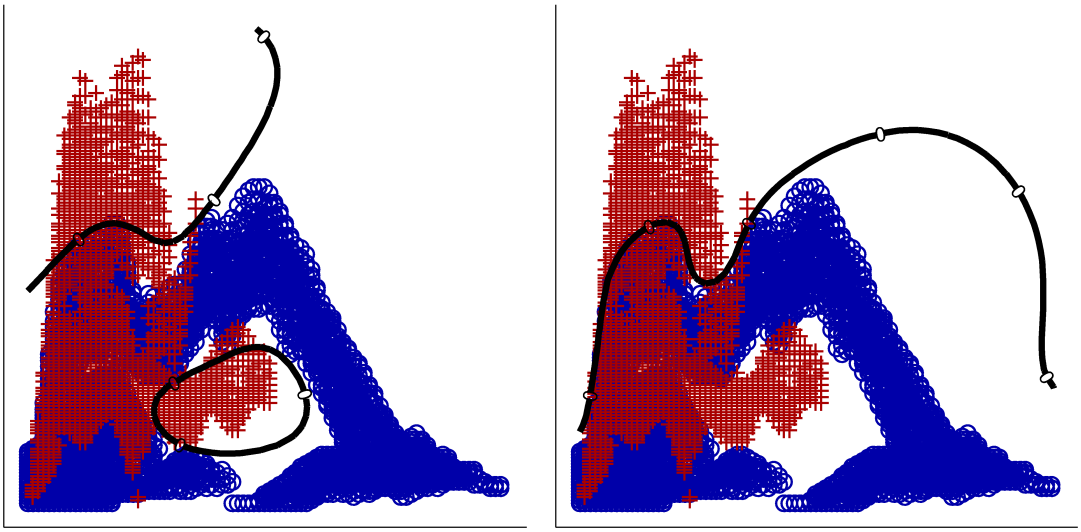
5.3 Data Filtering

Insular regions as depicted in Fig. 5.5(a) might be counterintuitive according to domain knowledge. Therefore, it becomes necessary to provide methods for avoiding such undesired results during the model building process. Such unintended results can be prevented by removing conflicting data points from the data set. In the following, we will show two heuristics, namely: (1) the convex hull filtering and (2) the upper envelope filtering. The basic idea behind both approaches is to remove data points which may conflict with monotony requirements in the resulting classification models. Furthermore, removing conflicting data points leads to a runtime reduction of the submodel learning process. Another method for data filtering might be the application of min-values as basic filter on single dimensions.

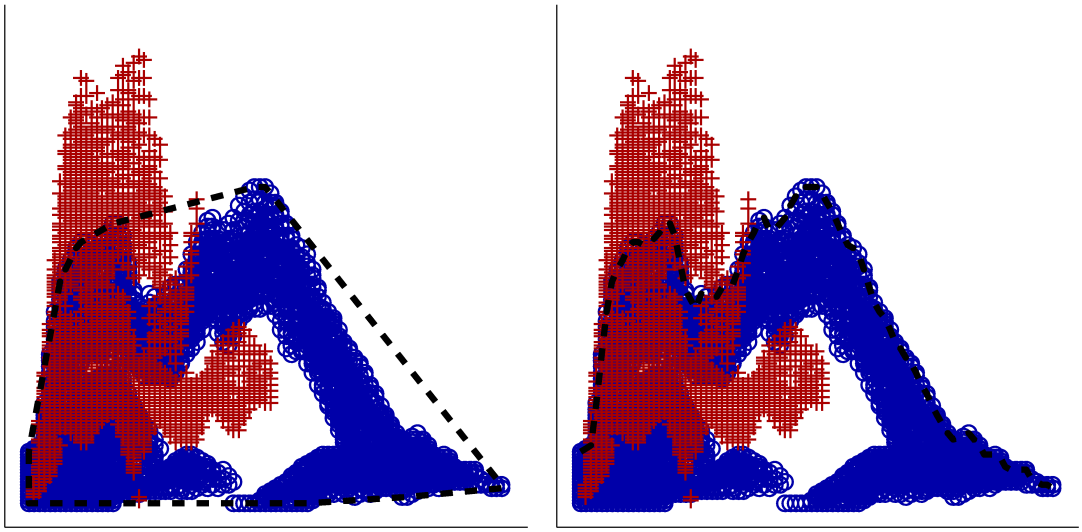
5.3.1 Convex Hull Filtering

This approach determines the convex hull of all samples belonging to the default class (c_{pref}). The convex hull of a set of data points is the minimal convex set containing all data points. Within our experiments we are using the Quickhull algorithm proposed by Barber et al. (1996) in order to determine the convex hull of the default class. All samples which do not belong to the default class and which are within the convex hull are removed from training data set. The classifier is then trained on the reduced data set.

The advantages of this approach are that the axes orientation does not influence the result of the algorithm and that it can be extended to an arbitrary dimensionality. As one can see in Fig. 5.5(c), the convex hull of a set of data points belonging to class ◯ can include regions of the other class which might be separable from the class ◯. Thus a more sensitive filtering method is needed.



(a) Unintended insular region of a submodel. (b) Submodel after applying the upper envelope filter.



(c) Convex hull filter. (d) Upper envelope filter.

Fig. 5.5: Convex hull filtering and upper envelope filtering. The filter is applied for class ◯. The decision borders of the resulting filter functions are plotted as dashed lines.

5.3.2 Upper Envelope Filtering

In this approach, the upper envelope[5] of all default class samples within a two-dimensional projection is determined and all samples which do not belong to the default class and which are below the upper envelope are removed from the training data set. Then, the classifier is trained on the reduced data set.

For this approach the axes orientation influences the result of the filter. This can be compensated by filtering both variants of a (two-dimensional) feature combination and presenting them both to the function that determines the best submodel. Another solution is to define a set of dimensions where a monotonic behavior is expected. For such dimensions one can choose the axes orientation directly.

This variant of filtering is much more sensitive than the convex hull filter, as one can see in Fig. 5.5(d), but it works only on two-dimensional data where one dimension has a (local) monotony constraint.

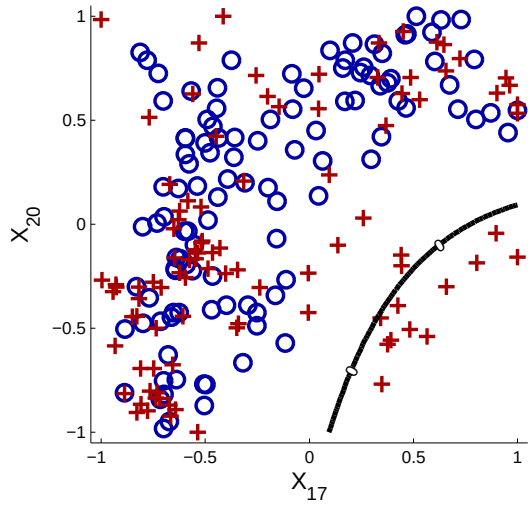
5.4 A Naval Risk Detection Example

This data set serves as an example of the advantages of using data filtering or feature construction methods in order to facilitate the low-dimensional submodels to solve a classification problem. The SONAR data set consists of 208 instances. Each instance is described by 60 attributes. The task is to discriminate between sonar signals bounced off a metal cylinder, that is possibly a mine, (CLASS 1) and those reflected by a roughly cylindrical rock (CLASS 0). This data set can be obtained from UCI Machine Learning Repository (Asuncion & Newman, 2007).

The submodels are trained as support vector machines with a Gaussian kernel. The default class is set to CLASS 1, $c_{pref} = 1$, that is, a metal cylinder must not be classified as a rock by any model. In order to reduce the risk of misclassifying any novel CLASS 1 sample the misclassification costs for CLASS 1 are 20 times higher than for CLASS 0.

This problem cannot be sufficiently solved by a Non-hierarchical Ensemble Model with respect to the restriction to two-dimensional projections since there is a large overlap of both classes within all possible two-dimensional projections. Using more sensitive model parameters might lead to a better performance but the submodels will capture only one or two samples at once. Thus numerous submodels will be built which increase the risk of overfitting and decrease the interpretability of the solution. The predictive performance of the Non-hierarchical Ensemble Model trained on the complete data set is given as confusion matrix in Fig. 5.6(b). This Non-hierarchical Ensemble Model consists of three submodels. Further submodels

[5] The upper envelope of a set of two-dimensional data points determines for each value on the X_1-axis the largest observed value on the X_2-axis.



(a) 1st submodel g_1 with $\beta_1 = \{17, 20\}$. This submodel captures 11 CLASS 0 samples.

	predicted class	
true class	CLASS 1	CLASS 0
CLASS 1	111	0
CLASS 0	63	34

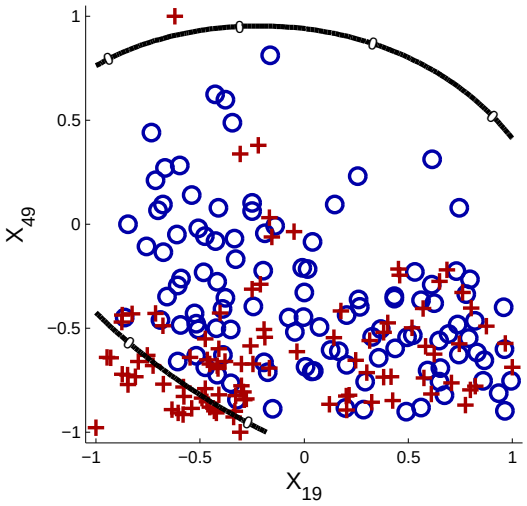
(b) Confusion matrix of the complete Non-hierarchical Ensemble Model.

Fig. 5.6: Non-hierarchical Ensemble Model and the SONAR data set trained without additional features. CLASS 1 samples are marked with circles and CLASS 0 samples are marked with crosses. The decision boundary is drawn as solid line.

cannot be determined. The best submodel is depicted in Fig. 5.6(a). A better predictive performance can be achieved by increasing the number of dimensions per submodel: allowing three-dimensional projections it becomes possible to solve this problem but the decision borders of the submodels will become more complex.

Another possibility to tackle this problem is to apply one of the data filtering methods which are discussed in Sect. 5.3. Applying such a data filtering method (for instance, the convex hull filtering) the submodels must only handle those data points that can actually be separated within the low-dimensional subspaces. Thus the disadvantageous effect of the conflicting data points can be reduced and the classification problem becomes simpler. The outcome of the application of the convex hull filtering method is illustrated in Fig. 5.7(a) and Fig. 5.7(b).

Another method that can be used to solve this classification problem is to apply the feature construction method of Sect. 5.2. Hereby and for this application example, 20 additional input dimensions are added to the original data set. The number of additional input dimensions can be chosen by the user. In this example, those hidden neurons of the MLP are selected that are most correlated with the target variable. This facilitates the Non-hierarchical Ensemble Model to completely solve this classification problem with 18 submodels. The corresponding confusion matrix is given in Fig. 5.8(b). Even if one regards only the first three submodels, which already capture 61 of 97 CLASS 0 samples, one achieves a higher predictive accuracy compared to the Non-hierarchical Ensemble Model without additional input dimensions and without a preceding data filtering. The best submodel of the Non-hierarchical Ensemble



(a) 1st submodel g_1 with $\beta_1 = \{19, 49\}$. This submodel captures 25 CLASS 0 samples.

	predicted class	
true class	CLASS 1	CLASS 0
CLASS 1	111	0
CLASS 0	5	92

(b) Confusion matrix of the complete Non-hierarchical Ensemble Model.

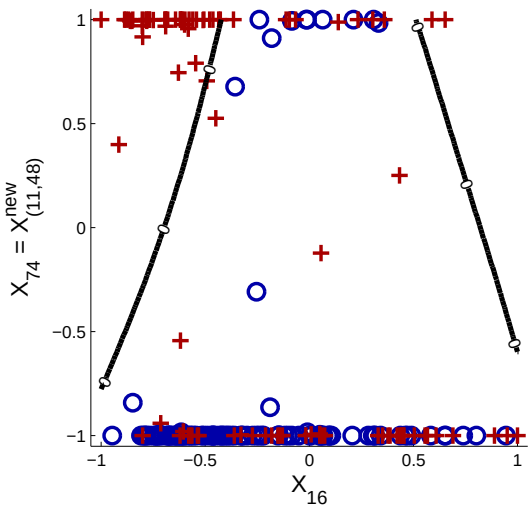
Fig. 5.7: Non-hierarchical Ensemble Model and the SONAR data set trained with preceding convex hull data filtering.

Model with additional input dimensions is depicted in Fig. 5.8(a). This submodel uses one original input dimension and one additionally generated feature. By applying the feature construction method, which is described in Sect. 5.2, the following additional input dimension $X_{74} = X^{\text{new}}_{(11,48)}$ has been generated:

$$X_{74} = \tanh\left(X_{11} \cdot w^{\text{hidden}}_{(11,48)} + X_{48} \cdot w^{\text{hidden}}_{(48,11)} + b^{\text{hidden}}_{(11,48)}\right) ,$$

where $w^{\text{hidden}}_{(11,48)} = -36.829$, $w^{\text{hidden}}_{(48,11)} = -9.004$ and $b^{\text{hidden}}_{(11,48)} = -28.936$. This additional feature can be seen as preceding linear soft classifier that facilitates the (two-dimensional) submodels of the Non-hierarchical Ensemble Model to solve the classification problem, while the model remains interpretable. In Fig. 5.9 the surface plot of the additional input dimension X_{74} and the resulting partitioning of the input space is depicted.

Tab. 5.1 illustrates the gain of predictive performance by applying our feature construction method. We compare the original SONAR data set with the SONAR (MLP) data set where 20 additionally generated input dimensions are appended to the original input dimensions. Therefore, it becomes possible to reduce the predictive error of the Non-hierarchical Ensemble Model model by approx. 50% by using additionally generated input dimensions as preceding soft classifiers.



(a) 1st submodel g_1 with $\beta_1 = \{16, 74\}$. This submodel captures 44 CLASS 0 samples.

	predicted class	
true class	CLASS 1	CLASS 0
CLASS 1	111	0
CLASS 0	0	97

(b) Confusion matrix of the complete Non-hierarchical Ensemble Model.

Fig. 5.8: Non-hierarchical Ensemble Model and the SONAR data set trained with additional features.

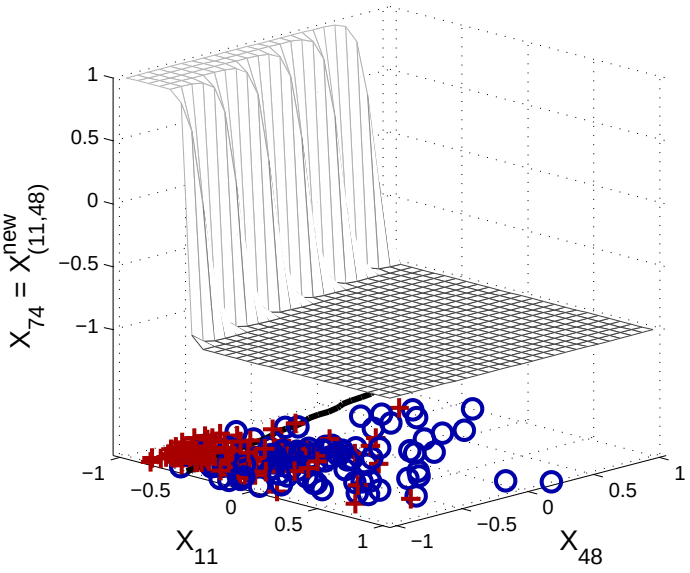


Fig. 5.9: Additionally generated input dimension $X_{74} = X^{\text{new}}_{(11,48)}$.

Tab. 5.1: SONAR data set: classification accuracy estimated by 10-fold crossvalidation with 10 random set initializations.

Method	#M	#D	$\widehat{err}$ mean (std)	$\widehat{err}_{crit}$ mean (std)
SONAR data set				
NHEM	6	2	36.35% (10.61)	*1.20%* (2.57)
libSVM	1	60	*11.10%* (7.64)	3.75% (4.46)
`treefit`	17	1	34.20% (11.09)	11.60% (7.78)
SONAR (MLP) data set				
NHEM	16	2	17.05% (8.82)	4.80% (4.82)
libSVM	1	80	*14.90%* (8.41)	*4.70%* (5.26)
`treefit`	15	1	36.65% (11.35)	11.25% (6.83)

5.5 Summary

The methods discussed within this chapter are important to achieve a good predictive performance of the learning algorithms which are given in Chap. 3 and Chap. 4. Feature selection is the most important issue since determining the best low-dimensional projection of the high-dimensional input space is the key idea of our ensemble methods. The best possible performance with the restriction to two- or three-dimensional submodels (that is on application problems where interpretability is imperative) can be achieved only by using the wrapper-based feature selection method that performs an exhaustive search through the search space of all possible combinations. Since this search space becomes very large for higher-dimensional problems another feature selection method may be used to reduce the set of relevant dimensions before using the wrapper to obtain the best projections. For application problems where interpretability is not essential one can also use the other feature selection methods that can determine all input dimensions that seem to be necessary to solve the learning problem. The use of those methods may increase the predictive performance since the learning algorithm can exploit more information within the submodels.

Another way to enhance the predictive performance of the submodels is to provide further input features that are (linear) combinations of the original input dimensions for the learning algorithm. Those methods become important for tasks, where the underlying problem cannot be divided into two- but only higher-dimensional subproblems. Unfortunately, increasing the predictive performance by feature construction methods decreases the interpretability of the learned model. Therefore, the key issue is to determine a good trade-off between both requirements.

The data filtering methods introduced in this chapter are useful for problems where it is known that there exist some ambiguously labeled data points or that the different classes have a strong overlap within the low-dimensional projections. Furthermore, the runtime of the learning algorithms used to build the submodels within Chap. 3 can be reduced due to the fact that the submodels are not forced to deal with ambiguous data.

6 Conclusions and Perspectives

The objective of this thesis is to provide a machine learning framework that is applicable in safety-related systems. Such systems are employed, for instance, in aerospace engineering, automotive industry, medical systems, or process automation. Within these domains it must be ensured that the system will not endanger the environment, technical equipment, or human life. It must be guaranteed that the system complies with all given requirements and that the solution is valid within the complete input space. In practical application tasks, the observed data used for learning the models is often scarce and the number of input dimensions is too large for sufficiently applying statistical risk estimation methods. The required error rates of a safety-related system are not grantable – especially for high-dimensional problems – by only assessing the error rates estimated on the basis of some testing data. Unfortunately, the given domain knowledge is often imperfect or the application problem is too complex in order to design an analytical solution of the problem. Therefore, it is necessary to combine both sources of information: the observed data and the existing domain knowledge. For successfully applying machine learning within the domain of safety-related applications it is imperative to provide an interpretable solution. It must be ensured that the extrapolation and interpolation behavior of the learned model is correct within the complete input space and meets all given functional requirements. Our approach discussed within this work is based on a data-driven model generation process allowing the domain experts to assess each single decision of the learned models.

6.1 Contributions of this Thesis

In the following we briefly summarize the main contributions of this thesis:

- A machine learning method for solving safety-related classification problems is provided in Chap. 3. This method is based on ensembles of low-dimensional submodels. The restricted dimensionality of the submodels ensures that the learned solution can be validated by the domain experts. Each submodel can be visualized facilitating the detection and avoidance of an unintended extra-

or interpolation behavior. Furthermore, the submodels can be interpreted independently (except for the DecisionTree-like Ensemble Model and its multiclass extension, where its is necessary to take the preceding nodes of the tree into account). The ensemble of the submodels compensates for the limited predictive performance of each single submodel. Our framework provides a good trade-off between (1) the interpretation and verification of the learned (sub-) models and (2) its predictive performance, while it is possible to guarantee that the solutions meet all given requirements.

- The binary classification ensemble is successfully adopted to solve the airbag deployment problem (cf. Sect. 3.4.1) having very high demands on the functional safety. The automated solution based on our algorithms replaces the current process where a manual calibration is necessary. Our solution significantly reduces the time of developing the control logic of the restraint systems. In addition, an improved performance with respect to crash detection and safety functions is achieved. Despite the sophisticated demands in the sensitive area of passenger safety, a verifiably correct solution can be provided by our algorithm.
- Our binary classification framework has been extended to also solve multiclass problems (cf. Sect. 3.3), while we are able to maintain the same desirable properties of the binary classification framework by introducing a hierarchy of misclassification costs which must be specified by the domain experts in order to rank the different classes according to the severity of their misclassification.
- A piecewise linear regression method (cf. Sect. 4.3) providing well-interpretable regression models has been proposed. The predictive performance of this approach is enhanced compared to other piecewise linear regression methods by the algorithm's capability of including information about the target variable into the process of partitioning the input space.
- An MLP-based feature construction (cf. Sect. 5.2) has been proposed in order to overcome possible limitations of the predictive performance that might arise due to the restricted dimensionality of the submodels within our ensemble methods. This method generates additional features that are low-dimensional linear combinations of the original input variables. These additional features can be seen as preceding soft classifiers enhancing the capability of the submodels to solve a learning problem that requires a higher dimensionality as allowed for the submodels.

6.2 Open Problems and Further Research

Feature selection is an important issue for applying our algorithms. Until now, we are mainly using the wrapper-based feature selection method in order to guarantee that the best low-dimensional submodels can be determined. Further research should be carried out in order to derive faster heuristics that still allow to find appropriate low-dimensional projections of the complete input space.

The Non-hierarchical Ensemble Model learning algorithm can be extended by a backtracking procedure in order to overcome possible local optima within the search space. Currently, this backtracking is performed during the interactive model selection procedure, where the domain experts can choose their favorite submodels. But within the automated model building procedures used for comparison in Appendix A we follow the greedy model selection scheme. Thus it might be possible to enhance the results that are listed in Appendix A.

The solution of regression problems within safety-related domains is challenging since it is not possible to distinguish between a correct and an incorrect decision – as within our classification framework, where the learning algorithms are forced to always correctly predict a certain group of data. Further research should be carried out with respect to combining submodels of different types and of different (data) sources (for instance, analytical models given by domain experts, data-driven generated models, look-up tables, etc.) as proposed by Beyer et al. (2008). This would allow the incorporation of prior knowledge in form of an analytical model providing a base-line performance and to learn models that can modify the prediction of the baseline model.

Appendix

A Experimental Evaluation on Common Benchmark Problems

A.1 Data Set Descriptions

Most classification data sets (except for the CUBES, CUBESMULT, and RESPIRATORY data sets) can be obtained from Asuncion & Newman (2007).

A.1.1 Binary Classification Problems

CUBES data set. The CUBES data set is generated from four Gaussian components in a five-dimensional space. For each CLASS 1 cluster 50 samples are drawn from $\mathsf{N}(e_n, 0.2 \cdot \mathbf{I})$, where e_n is a unit vector and $\mathbf{I}$ is the identity matrix. 100 samples of the CLASS 2 cluster are scattered around the origin, drawn from $\mathsf{N}((0,0,0)^\mathsf{T}, 0.2 \cdot \mathbf{I})$. CLASS 2 is chosen as default class $c_{\text{pref}} = 2$.

CUBES2 data set. The CUBES2 data set is a variation of the CUBES data set. In difference to this data set is that for each CLASS 1 cluster the samples are drawn from $\mathsf{N}(0.5 \cdot e_n, 0.2 \cdot \mathbf{I})$, where e_n is a unit vector and $\mathbf{I}$ is the identity matrix in order to obtain overlapping clusters.

HEPATITIS data set. The task of this data set is to predict whether a patient with hepatitis will die (CLASS 1) or survive (CLASS 2). CLASS 2 is chosen as the default class, $c_{\text{pref}} = 2$. The original data set from Asuncion & Newman (2007) consists of 155 instances and 20 attributes. There are lots of missing values. We are ignoring all dimensions with more than eight missing values. The instances of the resulting data set that still have missing values are removed from the data set. Hence, the final data set used within our experiments consists of 143 instances and 14 attributes.

MONKS-3 data set. The three MONKS problems were the basis of a general comparison of learning algorithms in Thrun et al. (1991). Here, the third problem is used, which can be described as: $f : a_1 \times a_2 \times a_3 \times a_4 \times a_5 \times a_6 \to \{0,1\}$, where $a_1 = \{1,2,3\}$, $a_2 = \{1,2,3\}$, $a_3 = \{1,2\}$, $a_4 = \{1,2,3\}$, $a_5 = \{1,2,3,4\}$, $a_6 = \{1,2\}$. The target function is defined as: $f := 1 \leftrightarrow (a_5 = 3 \wedge a_4 = 1) \vee (a_5 \neq 4 \wedge a_2 \neq 3)$. 5% class noise is added to the training set. The default class is set to $c_{\text{pref}} = 0$.

RESPIRATORY data set. This data set can be obtained from http://www.bangor.ac.uk/~mas00a/activities/real_data.htm. The data set consists of 85 clinical records (each 17 attributes) for premature newborn children with two types of respiratory distress syndrome: *Hyaline Membrane Disease* (CLASS 2) and *non-HMD* (CLASS 1). The two classes require immediate and completely different treatments, therefore an accurate classification is crucial within the first few hours after delivery. The default class is set to $c_{pref} = 2$.

SONAR and SONAR (MLP) data set. This data set includes 208 samples and 60 attributes. The task is to discriminate between sonar signals bounced off a roughly cylindrical rock (CLASS 0) and those bounced off a metal cylinder (a mine – CLASS 1). The default class is set to $c_{pref} = 1$ in order to avoid potential mines. The SONAR (MLP) data set is extended by 20 additionally generated attributes, which are returned by the MLP feature construction method, cf. Sect. 5.2.

WISCONSIN BREAST CANCER data set. This database consists of 699 instances and nine attributes. There are 16 missing attribute values – samples with missing values are ignored within our experiments. The task is to determine whether a sample is *benign* (CLASS 0) or *malignant* (CLASS 1). CLASS 1 is chosen as default class $c_{pref} = 1$.

A.1.2 Multi-Class Classification Problems

CUBESMULT data set. This data set is an extension of the CUBES data set to a four-class problem: the CLASS 2 samples are drawn from $N((0.0, 0.0, 0.0)^T, 0.2 \cdot \mathbf{I})$, the samples of CLASS 3 are drawn from $N((0.5, 0.5, 0.5)^T, 0.2 \cdot \mathbf{I})$, the CLASS 4 samples are drawn from $N((1.0, 1.0, 1.0)^T, 0.2 \cdot \mathbf{I})$, and the samples of CLASS 1 are drawn from $N(e_n + i \cdot 0.5, 0.2 \cdot \mathbf{I}), i = \{0, 1, 2\}$. The following hierarchy of misclassification costs is assumed: $penalty(\text{CLASS } 4) > penalty(\text{CLASS } 3) > penalty(\text{CLASS } 2) > penalty(\text{CLASS } 1)$.

DERMATOLOGY data set. This example data set is a challenging problem in dermatology Güvenir et al. (1998). The task is to discriminate six differential diagnostics of erythemato-squamous diseases, namely: PSORIASIS (CLASS 1), SEBOREIC DERMATITIS (CLASS 2), LICHEN PLANUS (CLASS 3), PITYRIASIS ROSEA (CLASS 4), CRONIC DERMATITIS (CLASS 5), and PITYRIASIS RUBRA PILARIS (CLASS 6). All these diseases share the clinical features of erythema and scaling – with minor differences. The data set consists of 366 records and each record has 33 attributes. The age attribute of the original data set from the UCI Machine Learning Repository (Asuncion & Newman, 2007) is omitted here, because is has some missing values. The following hierarchy of misclassification costs is assumed: $penalty(\text{CLASS } 6) > penalty(\text{CLASS } 5) > penalty(\text{CLASS } 4) > penalty(\text{CLASS } 3) > penalty(\text{CLASS } 2) > penalty(\text{CLASS } 1)$.

FISHER'S IRIS data set. This well-known data set from Fisher (1936) contains three classes (IRIS SETOSA – CLASS 1, IRIS VERSICOLOR – CLASS 2, and IRIS VIRGINICA – CLASS 3) of 50 instances each. The input space consists of four numeric attributes. CLASS 1 is linearly separable from the other two classes; CLASS 2 and CLASS 3 are not linearly separable from each other. The following hierarchy of misclassification costs is assumed: $penalty(\text{CLASS 3}) > penalty(\text{CLASS 1}) > penalty(\text{CLASS 2})$.

GLASS data set. The GLASS data set includes 214 instances and nine attributes. The task is to discriminate six different types of glass: CLASS 1 (BUILDING WINDOWS FLOAT PROCESSED), CLASS 2 (BUILDING WINDOWS NON FLOAT PROCESSED), CLASS 3 (VEHICLE WINDOWS FLOAT PROCESSED), CLASS 4 (CONTAINERS), CLASS 5 (TABLEWARE), and CLASS 6 (HEADLAMPS). The following hierarchy of misclassification costs is assumed: $penalty(\text{CLASS 1}) > penalty(\text{CLASS 2}) > penalty(\text{CLASS 3}) > penalty(\text{CLASS 4}) > penalty(\text{CLASS 5}) > penalty(\text{CLASS 6})$.

LYMPH data set. This data set consists of 148 instances and 19 attributes. It concerns a four-class problem. The classes are: LYMPH1 (*normal find*), LYMPH2 (*metastases*), LYMPH3 (*malign lymph*), and LYMPH4 (*fibrosis*). Within our experiments, the following hierarchy of misclassification costs is assumed: $penalty(\text{LYMPH2}) > penalty(\text{LYMPH3}) > penalty(\text{LYMPH4} > penalty(\text{LYMPH1})$.

NEWTHYROID data set. This problem concerns another typical medical data screening application. The classification task is to predict whether a patient's thyroid belongs to the class *euthyroidism* (NORMAL = CLASS 1), *hyperthyroidism* (HYPER = CLASS 2) or *hypothyroidism* (HYPO = CLASS 3). The data set consists of 215 records and each record is described by five attributes. The following hierarchy of misclassification costs is assumed: $penalty(\text{CLASS 3}) > penalty(\text{CLASS 2}) > penalty(\text{CLASS 1})$.

POST-OP data set. The classification task of this database is to determine where patients in a postoperative recovery area should be sent to next: PATIENT SENT TO INTENSIVE CARE UNIT – CLASS 1, PATIENT SENT TO GENERAL HOSPITAL FLOOR – CLASS 2, and PATIENT PREPARED TO GO HOME – CLASS 3. There are 90 instances and 8 attributes. The following hierarchy of misclassification costs is assumed: $penalty(\text{CLASS 1}) > penalty(\text{CLASS 2}) > penalty(\text{CLASS 3})$.

SEGMENTATION data set. The 2310 instances of this data set are drawn randomly from a database of seven outdoor images. Each instance is described by 19 continuous attributes. The task is to distinguish the following classes: CLASS 1 (BRICKFACE), CLASS 2 (SKY), CLASS 3 (FOLIAGE), CLASS 4 (CEMENT), CLASS 5 (WINDOW), and CLASS 6 (PATH). The following hierarchy of misclassification costs is assumed: $penalty(\text{CLASS 6}) >$

$penalty(\text{CLASS } 5) > penalty(\text{CLASS } 4) > penalty(\text{CLASS } 3) > penalty(\text{CLASS } 2) > penalty(\text{CLASS } 1)$.

WINE data set. This data set consists of the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. The analysis determined the quantities of 13 constituents found in each of the three types of wines. $penalty(\text{CLASS } 3) > penalty(\text{CLASS } 2) > penalty(\text{CLASS } 1)$ is assumed as hierarchy of misclassification costs.

A.1.3 Regression Problems

Within the experiments performed on the regression data sets, all input dimensions are normalized to the interval $[-1, 1]$ in order to rule out scaling effects.

ARTIFICIAL1 Data Set. We used the following function from Ferrari-Trecate & Muselli (2002) as target function:

$$f_1(x_1, x_2) = \begin{cases} 3 + 4x_1 + 2x_2 & \text{if} \quad A_1 \wedge A_3 \\ -5 - 6x_1 + 6x_2 & \text{if} \quad \neg A_1 \wedge \neg A_2 \\ -2 + 4x_1 - 2x_2 & \text{if} \quad A_2 \wedge \neg A_3 \end{cases},$$

where $A_1 : 0.5x_1 + 0.29x_2 \geqslant 0$, $A_2 : 0.5x_1 - 0.29x_2 \geqslant 0$, and $A_3 : x_2 \geqslant 0$. This target function is depicted in Fig. 4.1(a). 300 samples are drawn uniformly from $V^N = [-1, 1] \times [-1, 1]$ and y is determined as $y = f_1(x_1, x_2) + \epsilon$, where $\epsilon \sim N(0, 0.1)$.

ARTIFICIAL2 Data Set. This data set is used in Höppner & Klawonn (2003). The target function is $f_2(x_1, x_2) = \arctan(x_1)\cos({x_2}^2)$. This function is illustrated in Fig. 4.2(a). 300 data points are drawn uniformly from $V^N = [-2, 2] \times [-2, 2]$ and the value of the target y is determined by $y = f_2(x_1, x_2) + \epsilon$, where $\epsilon \sim N(0, 0.1)$.

AUTO MPG Data Set. This data set can be obtained from `http://www.liaad.up.pt/~ltorgo/Regression/DataSets.html`. The task of this data set is to predict the fuel consumption in miles per gallon (MPG) of different cars. The following five attributes of the original data set are used as input dimensions: 'acceleration', 'displacement', 'horsepower', 'model-year', 'weight'. The data set consists of 398 instances. Six instances with missing values are ignored within the experiments.

BODY FAT Data Set. This benchmark data set can be obtained from `http://lib.stat.cmu.edu/datasets/`. It includes 13 attributes and 252 instances. The task is to estimate the percentage of body fat based on different body circumference measurements.

Tab. A.1: Confusion matrix of a multi-class classifier (counts of data samples). The rows and columns are sorted according to the hierarchy of misclassification costs (cf. Sect. 3.3.1). The following ordering of the confusion matrix is assumed: $penalty(c_1) > penalty(c_2) > \ldots > penalty(c_k)$.

	predicted class			
true class	c_1	c_2	$\ldots$	c_K
c_1	$h_{1,1}$	$h_{1,2}$	$\ldots$	$h_{1,K}$
c_2	$h_{2,1}$	$h_{2,2}$	$\ldots$	$h_{2,K}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
c_K	$h_{K,1}$	$h_{K,2}$	$\ldots$	$h_{K,K}$

OZONE Data Set. The OZONE data set and the PROSTATE data set can be obtained from http://www-stat.stanford.edu/~tibs/ElemStatLearn/. The OZONE data set is used to estimate the daily ozone concentration based on three attributes: wind speed, daily maximum temperature, and solar radiation. This data set includes 111 instances.

PROSTATE Data Set. The task is to estimate the level of a prostate specific antigen based on a eight clinical measurements. The data set consists of 97 data points.

A.2 Experimental Setup

Classification. For every classifier one can obtain a confusion matrix as depicted in Tab. A.1. Based on those confusion matrices, we are computing three evaluation measures in order to compare the performance of the learned solutions. First, the overall *predictive error* of the model is estimated by:

$$\widehat{err} = \frac{1}{M}\left(M - \sum_{k=1}^{K} h_{k,k}\right),$$

where M denotes the number of all data points.

The second evaluation measure, the *critical error*, concerns only the samples that violate the predefined hierarchy of misclassification costs:

$$\widehat{err}_{crit} = \frac{1}{M}\sum_{i=1}^{K}\sum_{j=i+1}^{K} h_{i,j}\,.$$

The critical error is the more important error measurement because it corresponds to a violation of the given domain knowledge.

As third evaluation measure the *one-point-estimate of the area under the ROC curve* (AUC), cf. Ferri et al. (2003), is used:

$$\mathrm{AUC} = \max\left\{\frac{1}{K}, 1 - \frac{1}{K}\sum_{i=1}^{K}\sum_{j=1, i\neq j}^{K} n_{i,j}\right\},$$

where $n_{i,j} = h_{i,j}/\sum_{l=1}^{K} h_{i,l}$. In Appendix B you can find a brief introduction into the analyses of ROC curves.

Our classification ensemble methods are compared with a standard SVM implementation (libSVM) from Chang & Lin (2001) and a CART classification tree (`treefit` in Matlab). In Tab. A.2, the Non-hierarchical Ensemble Model is abbreviated as NHEM and the DecisionTree-like Ensemble Model is abbreviated as DTEM. Szepannek denotes a variation of the Non-hierarchical Ensemble Model, where for each submodel all input dimensions are retained that are returned by the Kolmogorov-Smirnoff test feature selection method. In Tab. A.3 the Hierarchical Separate-and-Conquer Ensemble is abbreviated as NHEM, the Ensemble of Multi-Class Submodels is abbreviated as DTEM, and the One-versus-Rest Ensemble is abbreviated as OvRE. The submodels of our ensemble models are SVMs with Gaussian kernels. The parameter sets of the SVMs are chosen manually in order to obtain smooth decision surfaces in the submodels. The same parameter sets are used for the high-dimensional SVM. For feature selection, our ensemble performs an exhaustive search through all possible pairs of features in order to determine the best low-dimensional projections.

The evaluation measures in Table A.2 and Table A.3 are estimated by a 10-fold-cross-validation procedure where the evaluation measures are estimated only on the testing data. For every data set we used 10 different fold initializations.

Regression. In the regression setting, that is in Tab. A.4, the performance of each learned model is estimated by the mean absolute error:

$$\widehat{err} = \frac{1}{M}\sum_{m=1}^{M} \left|y_m - \hat{f}(\vec{v}_m)\right|.$$

All data sets described in the following are randomly divided into a training data set (80% of the data) and a testing data set (20% of the data). 20 different pairs of training and testing data are generated for each data set.

Tab. A.2: 10-fold crossvalidation on binary classification problems, cf. Sect. A.1.1. The error is estimated on each testing fold and is averaged over 10 random fold initializations. For the experiments with the MLP feature construction method, the additionally generated features incorporate only pairs of the original input dimensions. #M denotes the number of submodels or decision nodes within the decision tree and #D denotes the dimensionality of each model or decision node, respectively.

Method	#M	#D	$\widehat{err}$ mean (std)	$\widehat{err}_{crit}$ mean (std)	AUC mean (std)
CUBES data set					
NHEM	3	2	2.00% (3.67)	0.21% (1.22)	0.983 (0.035)
Szepannek	1	3	36.79% (14.51)	0.00% (0.00)	0.741 (0.101)
DTEM	3	2	2.00% (3.67)	0.21% (1.22)	0.983 (0.035)
libSVM	1	5	1.64% (3.50)	0.14% (1.01)	0.988 (0.028)
`treefit`	10	1	25.43% (12.36)	6.00% (7.09)	0.769 (0.129)
CUBES2 data set					
NHEM	3	2	30.57% (13.28)	1.36% (2.99)	0.778 (0.085)
Szepannek	1	2	45.36% (13.02)	1.07% (3.11)	0.672 (0.091)
DTEM	3	2	30.00% (13.92)	1.14% (2.63)	0.783 (0.090)
libSVM	1	5	18.21% (10.52)	1.36% (2.99)	0.861 (0.078)
`treefit`	10	1	28.07% (10.80)	6.21% (6.23)	0.738 (0.115)
HEPATITIS data set					
NHEM	2	2	16.71% (9.26)	2.43% (4.79)	0.660 (0.150)
Szepannek	1	7	18.71% (8.84)	3.43% (5.12)	0.631 (0.136)
DTEM	5	2	19.43% (9.69)	6.29% (6.52)	0.668 (0.149)
libSVM	1	14	21.00% (10.09)	7.43% (6.33)	0.646 (0.150)
`treefit`	12	1	21.21% (9.33)	7.86% (7.90)	0.646 (0.134)
MONKS-3 data set					
NHEM	2	2	0.28% (1.59)	0.00% (0.00)	0.998 (0.014)
Szepannek	1	2	2.72% (2.50)	0.00% (0.00)	0.974 (0.023)
DTEM	2	2	0.28% (1.59)	0.00% (0.00)	0.998 (0.014)
libSVM	1	6	2.72% (2.50)	0.00% (0.00)	0.974 (0.023)
`treefit`	3	1	2.77% (2.35)	0.00% (0.00)	0.974 (0.022)
RESPIRATORY data set					
NHEM	4	2	10.13% (10.31)	4.88% (8.68)	0.898 (0.112)
Szepannek	1	10	11.13% (10.79)	6.13% (8.24)	0.891 (0.115)
DTEM	2	2	13.00% (11.08)	7.25% (7.98)	0.873 (0.117)
libSVM	1	17	9.88% (10.10)	5.75% (8.41)	0.899 (0.110)
`treefit`	5	1	19.00% (14.49)	6.00% (9.65)	0.813 (0.142)

Tab. A.2: 10-fold crossvalidation on binary classification problems (cont'd).

Method	#M	#D	$\widehat{err}$ mean (std)	$\widehat{err}_{crit}$ mean (std)	AUC mean (std)
			SONAR data set		
NHEM	7	2	36.30% (9.52)	1.10% (2.71)	0.611 (0.058)
Szepannek	1	33	13.15% (7.77)	5.00% (5.03)	0.863 (0.082)
DTEM	11	2	23.30% (8.88)	6.90% (5.94)	0.756 (0.096)
libSVM	1	60	10.85% (6.59)	3.45% (4.19)	0.886 (0.071)
`treefit`	16	1	28.95% (10.06)	10.35% (6.68)	0.705 (0.096)
			SONAR (MLP) data set		
NHEM	16	2	17.80% (8.66)	4.55% (4.50)	0.817 (0.090)
Szepannek	1	54	14.65% (7.53)	5.15% (5.24)	0.851 (0.074)
DTEM	10	2	17.50% (8.24)	6.40% (5.82)	0.822 (0.086)
libSVM	1	80	14.10% (7.43)	4.30% (4.72)	0.857 (0.073)
`treefit`	13	1	23.40% (8.99)	6.80% (5.44)	0.762 (0.093)
			WISCONSIN BREAST CANCER data set		
NHEM	4	2	5.96% (6.89)	0.93% (1.27)	0.949 (0.051)
Szepannek	1	9	4.85% (2.34)	2.26% (1.77)	0.947 (0.028)
DTEM	5	2	3.99% (2.26)	1.25% (1.44)	0.961 (0.024)
libSVM	1	9	4.85% (2.34)	2.26% (1.77)	0.947 (0.028)
`treefit`	41	1	6.84% (3.43)	2.32% (1.80)	0.932 (0.034)

Tab. A.3: 10-fold crossvalidation on multi-class classification problems, cf. Sect. A.1.2. The error is estimated on each testing fold and is averaged over 10 random fold initializations. The $\widehat{err}$ column denotes the overall classification error (including the missed samples of the One-versus-Rest Ensemble). #M denotes the number of submodels or decision nodes within the decision tree and #D denotes the dimensionality of each model or decision node, respectively.

Method	#M	#D	$\widehat{err}$ mean (std)	$\widehat{err}_{crit}$ mean (std)	AUC mean (std)	missed samples
			CUBESMULT data set			
HSCE	5	2	7.74% (4.09)	0.50% (1.09)	0.932 (0.056)	
Szepannek	3	5	32.19% (16.15)	0.33% (0.83)	0.881 (0.058)	
EMCS	7	2	3.74% (2.88)	1.38% (1.80)	0.951 (0.049)	
OvRE	6	2	34.31% (6.81)	0.26% (0.82)	0.975 (0.075)	[34.0%]
libSVM	1	5	1.88% (2.09)	0.29% (0.78)	0.987 (0.024)	
$\text{libSVM}_{(ho)}$	6	5	1.83% (2.05)	0.24% (0.72)	0.988 (0.023)	
`treefit`	28	1	19.62% (6.48)	5.02% (3.56)	0.817 (0.082)	

Tab. A.3: 10-fold crossvalidation on multi-class classification problems (cont'd).

Method	#M	#D	$\widehat{err}$ mean (std)	$\widehat{err}_{crit}$ mean (std)	AUC mean (std)	missed samples
			CUBES2MULT data set			
HSCE	6	2	29.36% (6.97)	0.69% (1.32)	0.850 (0.062)	
Szepannek	3	5	60.88% (17.80)	0.19% (0.65)	0.783 (0.060)	
EMCS	8	2	14.52% (5.50)	1.52% (1.93)	0.911 (0.059)	
OvRE	7	2	55.76% (7.47)	0.40% (0.90)	0.958 (0.094)	[55.4%]
libSVM	1	5	14.43% (5.47)	0.88% (1.34)	0.932 (0.038)	
libSVM$_{(ho)}$	6	5	14.29% (5.49)	0.71% (1.24)	0.936 (0.036)	
`treefit`	33	1	0.22.10% (5.68)	6.60% (4.06)	0.773 (0.096)	
			DERMATOLOGY data set			
HSCE	7	2	10.31% (5.27)	2.33% (2.15)	0.889 (0.056)	
Szepannek	5	15	5.42% (3.69)	1.06% (1.80)	0.940 (0.044)	
EMCS	10	2	5.56% (3.68)	3.17% (2.90)	0.927 (0.058)	
OvRE	12	2	24.83% (6.96)	0.36% (0.94)	0.970 (0.045)	[22.6%]
libSVM	1	33	3.39% (2.84)	1.94% (2.25)	0.963 (0.035)	
libSVM$_{(ho)}$	15	33	3.47% (2.91)	1.94% (2.25)	0.962 (0.036)	
`treefit`	11	1	5.64% (3.65)	1.47% (1.95)	0.944 (0.044)	
			FISHER'S IRIS data set			
HSCE	2	2	9.13% (8.46)	0.80% (2.73)	0.911 (0.079)	
Szepannek	2	4	3.60% (4.59)	0.27% (2.10)	0.963 (0.048)	
EMCS	1	2	6.73% (6.18)	4.07% (4.53)	0.931 (0.070)	
OvRE	3	2	15.67% (10.40)	0.80% (2.73)	0.981 (0.045)	[14.2%]
libSVM	1	4	4.20% (5.07)	0.87% (2.79)	0.958 (0.054)	
libSVM$_{(ho)}$	3	4	4.20% (5.07)	0.87% (2.79)	0.958 (0.054)	
`treefit`	12	1	8.93% (7.53)	4.13% (5.59)	0.914 (0.075)	
			GLASS data set			
HSCE	11	2	43.71% (11.25)	8.52% (5.43)	0.620 (0.115)	
Szepannek	4	5	37.19% (10.95)	7.57% (6.46)	0.686 (0.095)	
EMCS	16	2	32.86% (11.25)	13.43% (8.24)	0.671 (0.148)	
OvRE	15	2	77.62% (9.66)	6.81% (5.59)	0.775 (0.142)	[67.3%]
libSVM	1	9	30.90% (10.07)	14.24% (7.40)	0.721 (0.123)	
libSVM$_{(ho)}$	15	9	33.29% (9.70)	11.24% (6.82)	0.699 (0.113)	
`treefit`	28	1	48.67% (10.52)	7.19% (5.35)	0.465 (0.121)	

Tab. A.3: 10-fold crossvalidation on multi-class classification problems (cont'd).

Method	#M	#D	$\widehat{err}$ mean (std)	$\widehat{err}_{crit}$ mean (std)	AUC mean (std)	missed samples
			LYMPH data set			
HSCE	6	2	18.64% (9.52)	4.93% (5.89)	0.837 (0.131)	
Szepannek	3	7	18.00% (10.53)	5.71% (6.18)	0.838 (0.143)	
EMCS	9	2	17.43% (10.37)	6.79% (7.49)	0.814 (0.152)	
OvRE	16	2	36.29% (11.87)	4.00% (5.59)	0.876 (0.133)	[26.6%]
libSVM	1	18	22.07% (10.99)	4.50% (6.15)	0.768 (0.167)	
libSVM$_{(ho)}$	6	18	22.07% (10.99)	4.50% (6.15)	0.768 (0.167)	
`treefit`	15	1	29.29% (11.86)	6.00% (6.94)	0.764 (0.139)	
			NEWTHYROID data set			
HSCE	3	2	4.19% (5.03)	1.10% (2.86)	0.962 (0.063)	
Szepannek	2	5	4.10% (4.64)	1.71% (3.80)	0.955 (0.076)	
EMCS	3	2	4.00% (4.53)	2.57% (3.79)	0.939 (0.087)	
OvRE	4	2	6.38% (5.63)	0.90% (2.41)	0.975 (0.062)	[4.7%]
libSVM	1	5	4.62% (4.56)	1.90% (3.71)	0.949 (0.075)	
libSVM$_{(ho)}$	3	5	4.62% (4.56)	1.90% (3.71)	0.949 (0.075)	
`treefit`	6	1	4.29% (4.36)	1.29% (2.12)	0.957 (0.052)	
			POST-OP data set			
HSCE	4	2	32.89% (16.18)	2.00% (4.29)	0.620 (0.141)	
Szepannek	2	7	27.89% (15.11)	2.22% (4.47)	0.647 (0.146)	
EMCS	1	2	27.56% (14.60)	3.33% (6.22)	0.645 (0.149)	
OvRE	5	2	87.44% (10.67)	0.22% (1.56)	0.915 (0.139)	[83.2%]
libSVM	1	7	28.11% (14.25)	3.11% (6.33)	0.644 (0.145)	
libSVM$_{(ho)}$	3	7	28.11% (14.08)	3.11% (6.33)	0.644 (0.145)	
`treefit`	9	1	41.00% (18.05)	4.56% (6.90)	0.575 (0.129)	
			SEGMENTATION data set			
HSCE	12	2	12.76% (6.56)	5.24% (4.41)	0.876 (0.066)	
Szepannek	6	14	9.52% (5.98)	4.76% (4.33)	0.906 (0.063)	
EMCS	8	2	13.14% (7.42)	7.52% (5.59)	0.874 (0.074)	
OvRE	14	2	29.38% (10.18)	3.24% (4.05)	0.903 (0.085)	[22.6%]
libSVM	1	18	10.86% (6.13)	5.76% (4.45)	0.894 (0.060)	
libSVM$_{(ho)}$	21	18	10.81% (6.12)	5.14% (4.32)	0.893 (0.061)	
`treefit`	16	1	15.33% (7.70)	7.14% (5.80)	0.842 (0.088)	

Tab. A.3: 10-fold crossvalidation on multi-class classification problems (cont'd).

Method	#M	#D	$\widehat{err}$ mean (std)	$\widehat{err}_{crit}$ mean (std)	AUC mean (std)	missed samples
			WINE data set			
HSCE	4	2	6.71% (5.55)	0.24% (1.16)	0.944 (0.051)	
Szepannek	2	11	3.76% (4.39)	1.29% (2.72)	0.964 (0.045)	
EMCS	4	2	5.65% (5.60)	2.00% (3.85)	0.949 (0.051)	
OvRE	6	2	15.00% (8.90)	0.06% (0.59)	0.986 (0.029)	[13.3%]
libSVM	1	13	2.12% (3.40)	0.94% (2.17)	0.978 (0.039)	
libSVM$_{(ho)}$	3	13	2.12% (3.40)	0.94% (2.17)	0.978 (0.039)	
`treefit`	8	1	8.65% (6.45)	3.53% (4.95)	0.922 (0.068)	

Tab. A.4: Comparison of the model complexity of different regression methods. For details about the data sets see Sect. A.1.3. #M denotes the number of submodels or the number of decision nodes in the (unpruned) regression tree. For the LinEM the number of initial clusters (i.e. before cluster pruning) is given in parentheses. #D denotes the dimensionality of each (sub-)model or decision nodes within the regression tree.

Data set name	Regression method	#M	#D	$\widehat{err}$ (std) training	$\widehat{err}$ (std) testing
ARTIFICIAL1	LinEM (softmax)	3 (of 3)	2	0.79 (0.02)	0.77 (0.10)
	LinEM (crisp)	3 (of 3)	2	0.20 (0.03)	*0.19* (0.09)
	LLM	5	2	0.95 (0.06)	1.01 (0.11)
	`robustfit`	1	2	1.48 (0.04)	1.47 (0.15)
	`treefit`	41	1	0.34 (0.02)	0.70 (0.10)
ARTIFICIAL2	LinEM (softmax)	6 (of 10)	2	0.15 (0.01)	0.16 (0.02)
	LinEM (crisp)	6 (of 9)	2	0.16 (0.03)	0.18 (0.02)
	LLM	8	2	0.20 (0.01)	0.21 (0.03)
	`robustfit`	1	2	0.50 (0.01)	0.50 (0.05)
	`treefit`	46	1	0.08 (0.01)	*0.15* (0.01)
AUTO MPG	LinEM (softmax)	7 (of 10)	2	1.98 (0.07)	2.18 (0.21)
	LinEM (crisp)	2 (of 2)	2	2.07 (0.06)	2.16 (0.23)
	LLM	8	5	1.91 (0.05)	*2.08* (0.20)
	`robustfit`	1	5	2.56 (0.06)	2.65 (0.25)
	`treefit`	66	1	1.04 (0.08)	2.49 (0.28)

Tab. A.4: Comparison of the model complexity of different regression methods (cont'd).

Data set name	Regression method	#M	#D	$\widehat{err}$ (std) training	$\widehat{err}$ (std) testing
BODY FAT	LinEM (softmax)	1 (of 1)	2	3.58 (0.10)	3.71 (0.39)
	LinEM (crisp)	1 (of 1)	2	3.58 (0.10)	3.71 (0.39)
	LLM	1	13	3.68 (0.09)	3.87 (0.40)
	`robustfit`	1	13	3.39 (0.10)	*3.70* (0.41)
	`treefit`	39	1	1.76 (0.12)	5.02 (0.49)
OZONE	LinEM (softmax)	2 (of 2)	2	12.82 (1.20)	*14.02* (2.97)
	LinEM (crisp)	2 (of 2)	2	12.72 (0.81)	14.60 (3.31)
	LLM	6	3	11.83 (0.82)	14.23 (1.87)
	`robustfit`	1	3	15.15 (0.58)	15.93 (2.43)
	`treefit`	20	1	6.97 (0.76)	14.59 (2.99)
PROSTATE	LinEM (softmax)	3 (of 5)	2	0.48 (0.03)	0.63 (0.10)
	LinEM (crisp)	3 (of 6)	2	0.48 (0.03)	0.64 (0.10)
	LLM	1	8	0.58 (0.03)	0.64 (0.12)
	`robustfit`	1	8	0.50 (0.03)	*0.61* (0.11)
	`treefit`	15	1	0.35 (0.04)	0.81 (0.09)

A.3 Discussion

Classification. Within our experiments that are summarized in Tab. A.2 and Tab. A.3, our ensemble modelling approach provides a good trade-off between predictive accuracy and interpretability. On the binary classification problems, our ensemble models with the restriction to two-dimensional submodels are at least competitive on most tested data sets compared to a high-dimensional SVM solution – expect for the SONAR data set because this data set requires a higher dimensionality of the submodels. But this can be compensated by our MLP-based feature construction method. For almost all multi-class data sets the One-versus-Rest Ensemble achieves the best performance by regarding the critical error. On the other hand, the overall error of this approach is worse compared to all other methods. This poor performance is due to a large number of samples that are missed by the submodels and are assigned to the "other" class. Nevertheless, samples that are labeled as "unrecognized" might be acceptable for some application problems. The Hierarchical Separate-and-Conquer Ensemble approach provides a good trade-off between the predictive performance – on both, the overall error and the critical error – and the interpretation of the models compared to the SVM solution that achieves the least

overall error on all data sets but incorporates always the complete input space. The critical error of the Ensemble of Multi-Class Submodels approach is quite large in all experiments because the hierarchy of misclassification costs is ignored within our experiments. On the other hand, this ensemble approach is the only variant that does not require a hierarchy of misclassification costs to build a multi-class model. While interpreting the decision boundaries of a high-dimensional SVM is quite infeasible, all ensemble approaches allow a visualization of the submodels and, thus, facilitate the incorporation of domain knowledge via an interactive model selection process.

Regression. Our experiments are summarized in Tab. A.4. Especially for a small number of submodels ($J \leqslant 3$) the LinEM approach (that is restricted to two-dimensional submodels within the experiments) achieves a better or at least similar performance compared to the LLM approach that always regards the complete input space within its submodels. The LinEM approach places the prototypes to regions which are relevant to build local regression models with a small predictive error instead of placing the prototypes to regions with a high data density. The LLM approach can compensate this weakness for a higher number of prototypes – however, the larger the number of prototypes the more difficult the interpretation of the solution will be. For the benchmark data sets, the model complexity of the LinEM approach is smaller as for all other regression approaches – the regression trees become quite large, the linear regression model uses the complete input space, and the LLM model uses always J prototypes with N-dimensional models, while the cluster pruning strategy and the restriction to two-dimensional submodels of the LinEM approach facilitates the interpretability. If one compares the best results of all regression methods (cf. Tab. A.4), the LinEM method provides the best performance on two data sets – on the remaining data sets the performance is only slightly worse compared to the best approaches.

B Estimation of the Area Under the ROC Curve

The area under the ROC curve is used as performance measure in our experiments on classification problems (cf. Chap. 3 and Appendix A). This section is intended to give a brief survey of the field of ROC graph analysis. More detailed introductions into the analysis of ROC graphs can be found, for instance, in Provost & Fawcett (2001) and Fawcett (2006). The original ROC analysis is designed for binary classification problems, possible extensions for multi-class ROC analysis are given, for instance, by Ferri et al. (2003); Hand & Till (2001); Lachiche & Flach (2003).

B.1 The ROC Space

Concern the confusion matrix of a binary classifier as depicted in Tab. B.1: Samples that belong to the positive class (+) might be predicted as positive (h_{TP}) or as negative (h_{FN}) samples by the classifier. Correspondingly, samples that belong to the negative class (−) might be classified as positive (h_{FP}) or negative (h_{TN}) samples. Based on this possible outcome, two metrics are computed in order to evaluate the performance of the classification model. The first metric, the *true positive rate*, concerns the classifier's probability of correctly assigning positive samples to the positive class. The second metric is the *false positive rate*. This is the classifier's probability of erroneously predicting negative samples as positive samples. Based on the confusion matrix given in Tab. B.1, the true positive rate (TPR) can be estimated by

$$\mathrm{TPR} = \frac{h_{TP}}{h_{TP} + h_{FN}} \tag{B.1}$$

Tab. B.1: Confusion matrix of a binary classification model (count of data samples).

	predicted class	
true class	+	−
+	h_{TP}	h_{FN}
−	h_{FP}	h_{TN}

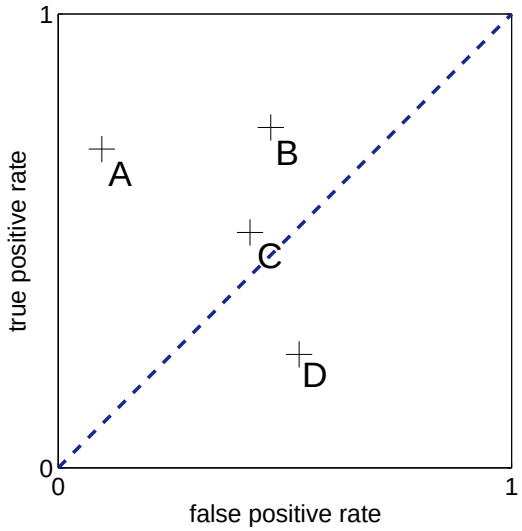


Fig. B.1: Four discrete classifiers within the ROC space.

and the false positive rate (FPR) can be estimated by

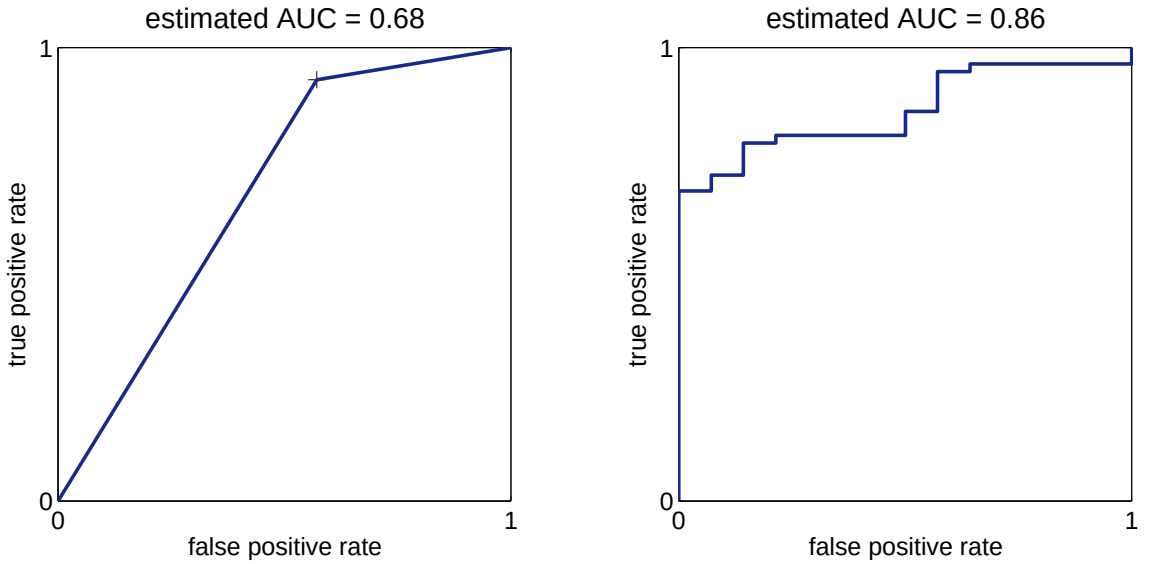
$$\mathrm{FPR} = \frac{h_{\mathrm{FP}}}{h_{\mathrm{FP}} + h_{\mathrm{TN}}}. \tag{B.2}$$

In Fig. B.1 the FPRs and the corresponding TPRs of four discrete classifiers (A, B, C and D) are shown. The resulting plot is called *receiver operating characteristics*[1] (ROC) graph and the space spanned by both metrics is called *ROC space*. In Fig. B.1, the point $(0,1)$ represents a perfect classifier, the point $(0,0)$ represents a classifier that always predicts the negative class, and the point $(1,1)$ represents a classifier that always predicts the positive class. The dashed line ($\mathrm{TPR} = \mathrm{FPR}$) represents the outcome of a binary classifier that randomly assigns the samples to the different classes. A classifier that lyes below this diagonal (that is classifier D in Fig. B.1) is worse than random guessing. Such a classifier can usually be negated in order to obtain a classifier that lyes beyond the diagonal and, thus, achieves a higher classification performance.

B.2 The Area Under The ROC Curve

The interesting question that arises in this setting is which of these classifiers performs best: the classifier B has the best TPR – the classifier A is only slightly worse but has a much smaller FPR. In order to combine both metrics and to obtain a single performance measure one can compute the *area under the ROC curve* (AUC). For

[1] The receiver operating characteristics is a classical methodology from signal detection Egan (1975).



(a) One-point estimate of the area under the ROC curve of a crisp classifier.

(b) Estimated area under the ROC curve of a soft classifier.

Fig. B.2: Receiver operating characteristic of two classification models. Both classifiers are trained on the HEPATITIS data set from Asuncion & Newman (2007).

a crisp classifier[2] the point $(\mathrm{FPR}, \mathrm{TPR})$ is connected with the points $(0,0)$ and $(1,1)$ and the area under this curve is computed. This is illustrated in Fig. B.2(a). For a soft classifier, the labels are sorted according to the score returned by the classifier. The corresponding values of the FPR and TPR per score level can be plotted as in Fig. B.2(b). In this setting, the AUC estimates the probability that, if we choose an example of the positive class and an example of the negative class, the classifier will give more score to the first one than to the second one.

In our experiments (cf. Chap. 3 and Appendix A), we are using for all classifiers the one-point estimate of the area under the ROC curve that is described in Hong et al. (2007). This is an pessimistic estimate of the soft AUC, but it can be computed for every tested classification method. For a binary classifier the one-point estimate of the area under a ROC curve can be computed as

$$\mathrm{AUC} = \frac{1 + \mathrm{TPR} - \mathrm{FPR}}{2}. \tag{B.3}$$

[2] A crisp classifier is a classifier that returns only the class label. In contrast to this, a soft classifier also returns a score or ranking to express its confidence about its prediction.

B.3 One-point Estimate of the AUC for Multi-Class Problems

Extending the analysis of the AUC to multi-class problems is non-trivial. A comparison of different multi-class AUC extensions is given in Ferri et al. (2003). For sake of simplicity, we are using the 1-point Trivial AUC Extension described in Ferri et al. (2003) for estimating the AUC. The values of the multi-class confusion matrix (cf. Tab. B.2) have to be normalized according to

$$n_{i,j} = \frac{h_{i,j}}{\sum_{k=1}^{K} h_{i,k}} \,. \tag{B.4}$$

Then, the AUC can be estimated by the average of the off-diagonal elements of the normalized confusion matrix, that is for a multi-class classifier the one-point estimate of the area under a ROC curve can be computed as

$$AUC = \max\left\{\frac{1}{K}, 1 - \frac{1}{K}\sum_{\substack{i=1}}^{K}\sum_{\substack{j=1\\ i\neq j}}^{K} n_{i,j}\right\} \,. \tag{B.5}$$

Note: For a binary classifier ($K = 2$) and with the restriction that a classifier is expected to perform better or equal than random guessing ($AUC \geqslant \frac{1}{2}$), this formulation is equivalent to the one-point AUC estimate of Eq. B.3:

$$\begin{aligned}
AUC &= \frac{1 + TPR - FPR}{2} \\
&= \frac{1 + \frac{h_{1,1}}{h_{1,1}+h_{1,2}} - \frac{h_{2,1}}{h_{2,1}+h_{2,2}}}{2} \\
&= \frac{1 + \left(1 - \frac{h_{1,2}}{h_{1,1}+h_{1,2}}\right) - \frac{h_{2,1}}{h_{2,1}+h_{2,2}}}{2} \\
&= \frac{2 - n_{1,2} - n_{2,1}}{2} \\
&= 1 - \frac{(n_{1,2} + n_{2,1})}{2} \\
&= 1 - \frac{1}{K}\sum_{\substack{i=1}}^{K}\sum_{\substack{j=1\\ i\neq j}}^{K} n_{i,j} \,.
\end{aligned}$$

Tab. B.2: Confusion matrix of a multi-class classifier (counts of data samples). If there is a certain hierarchy of misclassification costs (cf. Sect. 3.3.1) given for the multi-class problem, the following ordering of the confusion matrix is assumed: $penalty(c_1) > penalty(c_2) > \ldots > penalty(c_k) > \ldots > penalty(c_K)$.

	predicted class					
true class	c_1	c_2	$\ldots$	c_k	$\ldots$	c_K
c_1	$h_{1,1}$	$h_{1,2}$	$\ldots$	$h_{1,k}$	$\ldots$	$h_{1,K}$
c_2	$h_{2,1}$	$h_{2,2}$	$\ldots$	$h_{2,k}$	$\ldots$	$h_{2,K}$
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$		$\vdots$
c_k	$h_{k,1}$	$h_{k,2}$	$\ldots$	$h_{2,k}$	$\ldots$	$h_{k,K}$
$\vdots$	$\vdots$	$\vdots$		$\vdots$	$\ddots$	$\vdots$
c_K	$h_{K,1}$	$h_{K,2}$	$\ldots$	$h_{K,k}$	$\ldots$	$h_{K,K}$

C Notation

This appendix serves as quick reference of the symbols and notational conventions used within this thesis.

Sets and Random Variables

$\mathbb{N}$ denotes the set of natural numbers, $\mathbb{N} = \{1, 2, 3, \ldots\}$.

$\mathbb{R}$ denotes the set of real numbers.

$\mathbb{K}$ denotes the set of class labels, $\mathbb{K} = \{c_1, \ldots, c_k, \ldots, c_K\}$.

$X, Y, X_1, X_2, ..., X_N$ – random variables are denoted as uppercase letters.

$V^N = X_1 \times X_2 \times ... \times X_n \times ... \times X_N = \times_{n=1}^{N} X_n$ denotes the N-dimensional input space.

Y denotes the target variable of a learning problem – in a regression task $Y \subseteq \mathbb{R}$ and for a classification problem $Y \subseteq \mathbb{K}$.

$\mathbb{E}(X)$ denotes the the expected value of random variable X.

$\mathbb{E}(Y|X)$ denotes the the expected value of random variable Y given random variable X.

$x, x_m, x_m^{(n)}$ The observed values of random variables are denoted as lowercase letters. x_m denotes the m-th observation of the random variable X and $x_m^{(n)}$ denotes the m-th observation of the random variable X_n.

$\vec{v}_m$ denotes the m-th observation within the complete input space $\vec{v}_m \in V^N$, $\vec{v}_m = \left(x_m^{(1)}, x_m^{(2)}, \ldots x_m^{(N)}\right)$.

y_m denotes the target value of the m-th observation.

$\mathfrak{D} = \{(\vec{v}_1, y_1), \ldots, (\vec{v}_m, y_m), \ldots, (\vec{v}_M, y_M)\}$ denotes the set of the observed data, $\mathfrak{D} \subset V^N \times Y$.

Functions

$f : X_1 \times X_2 \times ... \times X_N \to Y$ denotes the true target function. This function is usually unknown.

$\hat{f} : V^N \to Y$ denotes the estimated target function given the observed data set $\mathfrak{D}$

$I(A)$ denotes the indicator function,

$$I(A) = \begin{cases} 1 & \text{if} \quad A \text{ is true} \\ 0 & \text{if} \quad A \text{ is false} \end{cases} .$$

$\pi_\beta\left(V^N\right) = V_\beta = \times_{n\in\beta} X_n$ denotes the projection of the original input space V^N to the subspace V_β, where $\beta \subset \{1, \ldots, N\}$.

$\text{sign}(x)$ denotes the sign function,

$$\text{sign}(x) = \begin{cases} 1 & \text{if} \quad x \geqslant 0 \\ -1 & \text{if} \quad x < 0 \end{cases} .$$

Vectors and Matrices

$\vec{a}$ denotes the vector

$$\vec{a} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} .$$

Within this thesis all vectors are assumed to be column vectors.

$\vec{a}^T$ denotes the transpose of the vector $\vec{a}$:

$$(a_1, a_2, a_3)^T = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix} .$$

$\|\vec{a}\|$ denotes the Euclidean norm of the vector $\vec{a} \in \mathbb{R}^N$

$$\begin{aligned} \|\vec{a}\| &= \sqrt{\vec{a}^T \vec{a}} \\ &= \sqrt{a_1^2 + a_2^2 + \ldots + a_N^2} \end{aligned}$$

e_n denotes the n-th unit vector within $\mathbb{R}^N$.

$$e_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \end{pmatrix}, e_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \\ \vdots \end{pmatrix}, e_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \\ \vdots \end{pmatrix}, \ldots$$

$\mathbf{I}$ denotes the identity matrix,

$$\mathbf{I} = \begin{bmatrix} 1 & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & 1 \end{bmatrix} .$$

$\mathbf{X}$ denotes a matrix $\mathbf{X} = (x_{i,j})$.
$\mathbf{X}^T$ denotes the transpose of matrix $\mathbf{X}$
$\mathbf{X}^{-1}$ denotes the inverse of matrix $\mathbf{X}$

$$\mathbf{X}^{-1}\mathbf{X} = \mathbf{X}\mathbf{X}^{-1} = \mathbf{I}\,.$$

Probability Distributions and Measures

$P(X = x)$ denotes the probability that random variable X takes value x. This is estimated by:

$$P(X = x) = \frac{\text{number of times value } x \text{ has occurred}}{\text{number of times the experiment was run}}$$

$P(X = x|Y = y)$ denotes the conditional probability that random variable X takes value x given $Y = y$. This probability is determined by:

$$P(X = x|Y = y) = \frac{P(X = x \cap Y = y)}{P(Y = y)}$$

$\mathrm{CDF}(X)$ denotes the cumulative probability distribution of random variable X, $\mathrm{CDF}(X) = P(X \leqslant x)$.

$\mathrm{CDF}(X|c_k)$ denotes the conditional cumulative probability distribution of random variable X given class c_k, that is $\mathrm{CDF}(X) = P(X \leqslant x|\, c_k)$.

$\mathrm{mean}(X)$ denotes the mean value of the random variable X. It is estimated by

$$\mathrm{mean}(X) = \frac{1}{M}\sum_{m=1}^{M} x_m\,,$$

where x_m is the m-th observation of random variable X. This value is a measure of the location of a random variable.

$\mathrm{std}(X)$ denotes the standard deviation of the random variable X. It is estimated by

$$\mathrm{std}(X) = \sqrt{\frac{1}{M-1}\sum_{m=1}^{M}(x_m - \mathrm{mean}(X))^2}\,.$$

This value is a measure of the spread around the location of a random variable.

$N(\mu, \sigma)$ denotes the normal distribution (also known as Gaussian distribution) with parameter $-\infty < \mu < \infty$ and $\sigma > 0$.

Classification Models

$f : V^N \to \mathbb{K}$ denotes a classification function that maps every data point $\vec{v} \in V^N$ to a certain class label $c_k \in \mathbb{K}$, where $\mathbb{K} = \{c_1, \ldots, c_k, \ldots, c_K\}$.

$g_j : \pi_{\beta_j}(V^N) \to \mathbb{K}$ denotes the j-th classification submodel with the projection $\pi_{\beta_j}(\cdot)$.

c_{pref} denotes the default class of a binary classification problem. By definition, this class must not be misclassified by any classification model.

$penalty(c_k)$ denotes the misclassification costs for class c_k. The hierarchy of misclassification costs defines an ordering of all class labels, cf. Eq. 3.6 on page 37.

Regression Models

$f : V^N \to \mathbb{R}$ denotes a regression function that assigns for every data point $\vec{v} \in V^N$ to a target value $y \in \mathbb{R}$.

$C(m) = j$ denotes the cluster mapping that assigns the data point $\vec{v}_m$ to the j-th cluster, with $1 \leqslant m \leqslant M$, $1 \leqslant j \leqslant J$, and $J < M$.

$\vec{p}_j \in V^N$ denotes the prototype vector of the j-th cluster.

$g_j : \pi_{\beta_j}(V^N) \to \mathbb{R}$ denotes the regression submodel that belongs to the j-th cluster, $C(m) = j$, and the projection $\pi_{\beta_j}(\cdot)$.

Performance Measures

Predictive error. The predictive error of a regression problem is estimated by:

$$\widehat{err} = \frac{1}{M} \sum_{m=1}^{M} \left| y_m - \hat{f}(\vec{v}_m) \right|$$

and for a classification problem the predictive error is estimated by:

$$\widehat{err} = \frac{1}{M} \left(M - \sum_{k=1}^{K} h_{k,k} \right)$$

given the confusion matrix in Tab. B.2 on page 109.

Critical error. The critical error denotes the relative number of samples, where the learned classification model violates the predefined hierarchy of misclassification costs. It is estimated by:

$$\widehat{err}_{crit} = \frac{1}{M} \sum_{i=1}^{K} \sum_{j=i+1}^{K} h_{i,j}$$

given an ordered confusion matrix as in Tab. B.2 on page 109 with $\forall i, j, i < j : penalty(c_i) > penalty(c_j)$.

TPR denotes the true positive rate, cf. Eq. B.1 in Appendix B.

FPR denotes the false positive rate, cf. Eq. B.2 in Appendix B.

AUC denotes the area under the ROC curve, cf. Eq. B.5 in Appendix B.

ROC denotes the receiver operating characteristics, cf. Appendix B

List of Figures

List of Tables

List of Algorithms

Bibliography

Aizerman, M. A., Braverman, E. M., & Rozonoer, L. I. (1964). Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25, 821–837.

Andrews, R., Diederich, J., & Tickle, A. B. (1995). Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge Based Systems*, 8(6), 373–389.

Asuncion, A. & Newman, D. J. (2007). UCI Machine Learning Repository. Available at http://www.ics.uci.edu/~mlearn/MLRepository.html.

Barber, C. B., Dobkin, D. P., & Huhdanpaa, H. (1996). The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4), 469–483.

Bedford, D. F., Austin, J., & Morgan, G. (1996). Requirements for a standard certifying the use of artificial neural networks in safety critical applications.

Beyer, J., Heesche, K., Hauptmann, W., & Otte, C. (2008). Combined knowledge-based and data-driven modeling by heterogeneous mixture-of-experts. In R. Mikut & M. Reischl (Eds.), *Proceedings of 18th Workshop Computational Intelligence* (pp. 204–213). Dortmund, Germany: Universitätsverlag Karlsruhe.

Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

Borgelt, C. & Kruse, R. (1998). Attributauswahlmaße für die Induktion von entscheidungsbäumen: Ein überblick. In G. Nakhaeizadeh (Ed.), *Data Mining: Theoretische Aspekte und Anwendungen* (pp. 77–98). Heidelberg, Germany: Physica-Verlag. in German.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of 5th Annual Workshop on Computational Learning Theory* (pp. 144–152). Pittsburgh, PA, USA: ACM Press.

Bowen, J. P. & Hinchey, M. G. (1999). *High-Integrity System Specification and Design*. Springer.

Boz, O. (2002). Extracting decision trees from trained neural networks. In *Proceedings of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining* (pp. 456–461). Edmonton, Alberta, Canada.

Breiman, L. (1996). Technical note: Some properties of splitting criteria. *Machine Learning*, 24, 41–47.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and Regression Trees*. CRC Press.

Brodley, C. E. & Utgoff, P. E. (1995). Multivariate decision trees. *Machine Learning*, 19(1), 45–77.

Cardoso, J. S., da Costa, J. F. P., & Cardoso, M. J. (2005). Modelling ordinal relations with SVMs: An application to objective aesthetic evaluation of breast cancer conservative treatment. *Neural Networks*, 18(5-6), 808–817.

Chang, C.-C. & Lin, C.-J. (2001). *LIBSVM: A library for support vector machines*. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

Chen, Y. & Wang, J. Z. (2003). Support vector learning for fuzzy rule-based classification systems. *IEEE Transactions on Fuzzy Systems*, 11(6), 716–728.

Cohen, W. W. (1995). Fast effective rule induction. In A. Prieditis & S. J. Russell (Eds.), *Proceedings of 12th International Conference on Machine Learning* (pp. 115–123). Tahoe City, CA, USA.

Cohn, D., Atlas, L., & Ladner, R. (1994). Improving generalization with active learning. *Machine Learning*, 15(2), 201–221.

Cortes, C. & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3), 273–297.

DEFSTAN 00-56 (2007). Defence standard 00-56/issue 4. Ministry of Defence, United Kingdom. Safety Management Requirements for Defence Systems.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1), 1–38.

Dietterich, T. G. (2000). Ensemble methods in machine learning. In *Proceedings of the First International Workshop on Multiple Classifier Systems* (pp. 1–15).: Springer.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern Classification*. Wiley, 2nd edition.

Egan, J. P. (1975). *Signal Detection Theory and ROC Analysis*. Series in Cognition and Perception. Academic Press.

Fawcett, T. (2006). An introduction to roc analysis. *Pattern Recognition Letters*, 27(8), 861–874.

Feller, W. (1948). On the Kolmogorov-Smirnov limit theorems for empirical distributions. *The Annals of Mathematical Statistics*, 19(2), 177–189.

Ferrari-Trecate, G. & Muselli, M. (2002). A new learning method for piecewise linear regression. In *ICANN '02: Proceedings of the International Conference on Artificial Neural Networks* (pp. 444–449). London, United Kingdom: Springer.

Ferri, C., Hernández-Orallo, J., & Salido, M. A. (2003). Volume under the ROC surface for multi-class problems. In *Proceedings of 14th European Conference on Machine Learning* (pp. 108–120). Dubrovnik, Croatia: Springer.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7, 179–188.

Frank, E. & Hall, M. (2001). A simple approach to ordinal classification. In *Proceedings of 12th European Conference on Machine Learning*, volume 2167 of *Lecture Notes In Computer Science* (pp. 145–156). Freiburg, Germany: Springer.

Friedman, J. H. (1996). *Another approach to polychotomous classification*. Technical report, Department of Statistics, Stanford University.

Friedman, J. H. & Tukey, J. W. (1974). A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, 23(9), 881–890.

Fürnkranz, J. (1999). Separate-and-conquer rule learning. *Artificial Intelligence Review*, 13(1), 3–54.

Gama, J. (2000). *Combining Classification Algorithms*. PhD thesis, University of Porto.

Gama, J. & Brazdil, P. (2000). Cascade generalization. *Machine Learning*, 41(3), 315–343.

Grabczewski, K. & Jankowski, N. (2005). Feature selection with decision tree criterion. In N. Nedjah, L. de Macedo Mourelle, A. Abraham, & M. Köppen (Eds.), *Proceedings of 5th International Conference on Hybrid Intelligent Systems* (pp. 212–217). Rio de Janeiro, Brazil: IEEE Computer Society.

Güvenir, H. A., Demiröz, G., & Ilter, N. (1998). Learning differential diagnosis of erythemato-squamous diseases using voting feature intervals. *Artificial Intelligence in Medicine*, 13(3), 147–165.

Guyon, I. & Elisseeff, A. (2006). An introduction to feature extraction. In I. Guyon, S. Gunn, M. Nikravesh, & L. Zadeh (Eds.), *Feature Extraction: Foundations and Applications* (pp. 1–25). Springer.

Guyon, I., Gunn, S., Nikravesh, M., & Zadeh, L., Eds. (2006). *Feature Extraction, Foundations and Applications*. Studies in Fuzziness and Soft Computing. Springer.

Hand, D. J. & Till, R. J. (2001). A simple generalisation of the area under the roc curve for multiple class classification problems. *Machine Learning*, 45(2), 171–186.

Hartman, E. J., Keeler, J. D., & Kowalski, J. M. (1990). Layered neural networks with gaussian hidden units as universal approximations. *Neural Computation*, 2(2), 210–215.

Hastie, T. & Stuetzle, W. (1989). Principal curves. *Journal of the American Statistical Association*, 84(406), 502–516.

Hastie, T. & Tibshirani, R. (1990). *Generalized Additive Models*. Chapman & Hall.

Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The Elements of Statistical Learning*. Springer.

Hathaway, R. J. & Bezdek, J. C. (1993). Switching regression models and fuzzy clustering. *IEEE Transactions on Fuzzy Systems*, 1(3), 195–204.

Hong, X., Chen, S., & Harris, C. J. (2007). A kernel-based two-class classifier for imbalanced data sets. *IEEE Transactions on Neural Networks*, 18(1), 28–41.

Höppner, F. & Klawonn, F. (2003). Improved fuzzy partitions for fuzzy regression models. *International Journal of Approximate Reasoning*, 32(2-3), 85–102.

Hsu, C.-W. & Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2), 415–425.

Huber, P. J. (1985). Projection pursuit. *The Annals of Statistics*, 13(2), 435–475.

IEC-61508 (2005). IEC/TR 61508: Functional Safety of Electrical / Electronic / Programmable Electronic Safety-Related Systems. International Electrotechnical Commission, Geneva.

Isaksen, U., Bowen, J., & Nissanke, N. (1997). *System and Software Safety in Critical Systems*. Technical Report RUCS/97/TR/062/A, Department of Computer Science, The University of Reading.

Kohavi, R. (1995a). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of 14th International Joint Conference on Artificial Intelligence*, volume 2 (pp. 1137–1143). Montreal, Quebec, Canada: Morgan Kaufmann.

Kohavi, R. (1995b). *Wrappers for Performance Enhancement and Oblivious Decision Graphs*. PhD thesis, Stanford University, Stanford, CA.

Kohavi, R. & John, G. H. (1997). Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2), 273–324.

Kohonen, T. (1990). The self-organizing map. *Proceedings of the IEEE*, 78(9), 1464–1480.

Kolman, E. & Margaliot, M. (2005). Are artificial neural networks white boxes? *IEEE Transactions on Neural Networks*, 16(4), 844–852.

Kuncheva, L. I. (2004). *Combining Pattern Classifiers: Methods and Algorithms*. Wiley.

Kurd, Z. & Kelly, T. (2007). Using fuzzy self-organising maps for safety critical systems. *Reliability Engineering & System Safety*, 92(11), 1563–1583.

Kurd, Z., Kelly, T., & Austin, J. (2006). Developing artificial neural networks for safety critical systems. *Neural Computing & Applications*, 16(1), 11–19.

Lachiche, N. & Flach, P. A. (2003). Improving accuracy and cost of two-class and multi-class probabilistic classifiers using roc curves. In T. Fawcett & N. Mishra (Eds.), *Proceedings of 20th International Conference on Machine Learning* (pp. 416–423). Washington, DC, USA: AAAI Press.

Lavrac, N. & Dzeroski, S. (1994). *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood.

Leveson, N. G. & Turner, C. S. (1993). An investigation of the Therac-25 accidents. *Computer*, 26(7), 18–41.

Lisboa, P. J. G. (2001). *Industrial use of safety-related artificial neural networks*. Contract research report 327/2001, Liverpool John Moores University.

Liu, H. & Setiono, R. (1998). Feature transformation and multivariate decision tree induction. In *Proceedings of the 1st International Conference on Discovery Science* (pp. 279 – 290). Fukuoka, Japan.

MacKay, D. J. C. (1992). *Bayesian methods for adaptive models*. PhD thesis, California Institute of Technology, Pasadena.

MacKay, D. J. C. (1998). Introduction to Gaussian processes. In C. M. Bishop (Ed.), *Neural Networks and Machine Learning* (pp. 133–166). Kluwer Academic Press.

MacQueen, J. B. (1967). Some methods for classification and analysis of multivariate observations. In L. M. LeCam & J. Neyman (Eds.), *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability* (pp. 281–297).: University of California Press.

Milton, J. S. & Arnold, J. C. (2002). *Introduction to Probability and Statistics: Principles and Applications for Engineering and the Computing Sciences*. McGraw-Hill, 4th edition.

MISRA (1994). Development guidelines for vehicle based software. MISRA - The Motor Industry Software Reliability Association.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Morgan, J. N. & Sonquist, J. A. (1963). Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association*, 58, 415–34.

Murthy, S. K., Kasif, S., & Salzberg, S. (1994). A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2, 1–32.

Nabney, I. (2002). *NETLAB: Algorithms for pattern recognition*. Springer.

Nauck, D. (2003). Fuzzy data analysis with NEFCLASS. *International Journal of Approximate Reasoning*, 32(2-3), 103–130.

Nauck, D., Klawonn, F., & Kruse, R. (1997). *Foundations of Neuro-Fuzzy Systems*. Wiley.

Nauck, D. & Kruse, R. (1999). Obtaining interpretable fuzzy classification rules from medical data. *Artificial Intelligence in Medicine*, 16(2), 149–169.

Nusser, S., Otte, C., & Hauptmann, W. (2007). Learning binary classifiers for applications in safety-related domains. In R. Mikut & M. Reischl (Eds.), *Proceedings of 17th Workshop Computational Intelligence* (pp. 139–151). Dortmund, Germany: Universitätsverlag Karlsruhe.

Nusser, S., Otte, C., & Hauptmann, W. (2008a). An EM-based piecewise linear regression algorithm. In E. Corchado, A. Abraham, & W. Pedrycz (Eds.), *Proceedings of 3rd International Workshop on Hybrid Artificial Intelligence Systems*, volume 5271 of *Lecture Notes in Artificial Intelligence* (pp. 466–474). Burgos, Spain: Springer.

Nusser, S., Otte, C., & Hauptmann, W. (2008b). Interpretable ensembles of local models for safety-related applications. In M. Verleysen (Ed.), *Proceedings of 16th European Symposium on Artificial Neural Networks* (pp. 301–306). Brugge, Belgium: D-facto publications.

Nusser, S., Otte, C., & Hauptmann, W. (2008c). Multi-class modeling with ensembles of local models for imbalanced misclassification costs. In O. Okun & G. Valentini (Eds.), *Proceedings of 2nd Workshop on Supervised and Unsupervised Ensemble Methods and Their Applications (at ECAI 2008)* (pp. 36–40). Patras, Greece.

Nusser, S., Otte, C., & Hauptmann, W. (2009a). Multi-class extension of verifiable ensemble models for safety-related applications. In A. Fink, B. Lausen, W. Seidel, & A. Ultsch (Eds.), *Advances in Data Analysis, Data Handling and Business Intelligence*, Studies in Classification, Data Analysis, and Knowledge Organization (pp. 733–744).: Springer.

Nusser, S., Otte, C., & Hauptmann, W. (2009b). Verifiable ensembles of low-dimensional submodels for multi-class problems with imbalanced misclassification costs. In O. Okun & G. Valentini (Eds.), *Applications of Supervised and Unsupervised Ensemble Methods*, volume 245 of *Studies in Computational Intelligence* (pp. 191–211). Springer.

Nusser, S., Otte, C., Hauptmann, W., & Kruse, R. (2009c). Learning verifiable ensembles for classification problems with high safety requirements. In L. S. L. Wang & T.-P. Hong (Eds.), *Intelligent Soft Computation and Evolving Data Mining: Integrating Advanced Technology*. IGI Global. accepted.

Nusser, S., Otte, C., Hauptmann, W., Leirich, O., Krätschmer, M., & Kruse, R. (2009d). Maschinelles Lernen von validierbaren Klassifikatoren zur autonomen Steuerung sicherheitsrelevanter Systeme. *at – Automatisierungstechnik*, 57(3), 138–145. in German.

Otte, C., Nusser, S., & Hauptmann, W. (2006). Machine learning methods for safety-related domains: Status and perspectives. In *Proceedings of Symposium on Fuzzy Systems in Computer Science* (pp. 139–148). Magdeburg, Germany.

Provost, F. & Fawcett, T. (2001). Robust classification for imprecise environments. *Machine Learning*, 42(3), 203–231.

Pullum, L. L., Taylor, B. J., & Darrah, M. A. (2007). *Guidance for the Verification and Validation of Neural Networks*. Wiley.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Rasmussen, C. E. & Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press.

Rehm, F., Klawonn, F., & Kruse, R. (2006). POLARMAP - a new approach to visualisation of high dimensional data. In *Proceedings of the 10th International Conference on Information Visualisation (IV'06)* (pp. 731–740). London, United Kingdom.

Ritter, H. (1991). Learning with the self-organizing map. In T. Kohonen, K. Mäkisara, O. Simula, & J. Kangas (Eds.), *Artificial Neural Networks* (pp. 379–384). Amsterdam, Netherlands: Elsevier.

Schlang, M., Feldkeller, B., Lang, B., Poppe, T., & Runkler, T. (1999). Neural computation in steel industry. In *Proceedings of European Control Conference '99* (pp. 1–6). Karlsruhe, Germany: Verlag rubicon.

Schölkopf, B. & Smola, A. (2002). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press.

Stone, C. J. (1985). Additive regression and other nonparametric models. *The Annals of Statistics*, 13(2), 689–705.

Szepannek, G. & Weihs, C. (2006). Local modelling in classification on different feature subspaces. In P. Perner (Ed.), *Proceedings of 6th Industrial Conference on Data Mining*, Lecture Notes in Artificial Intelligence (pp. 226–238). Leipzig, Germany: Springer.

Tax, D. M. J. & Duin, R. P. W. (2002). Using two-class classifiers for multiclass classification. In *Proceedings of the 16th International Conference on Pattern Recognition*, volume 2 (pp. 124–127). Quebec, Canada.

Taylor, B. J., Ed. (2005). *Methods and Procedures for the Verification and Validation of Artificial Neural Networks*. Springer.

Thorburn, W. M. (1918). The myth of occam's razor. *Mind*, 27(3), 345–353.

Thrun, S., Bala, J., Bloedorn, E., Bratko, I., Cestnik, B., Cheng, J., Jong, K. D., Dzeroski, S., Hamann, R., Kaufman, K., Keller, S., Kononenko, I., Kreuziger, J., Michalski, R., Mitchell, T., Pachowicz, P., Roger, B., Vafaie, H., de Velde, W. V., Wenzel, W., Wnek, J., & Zhang, J. (1991). *The MONK's Problems: A Performance Comparison of Different Learning Algorithms*. Technical Report CMU-CS-91-197, Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, USA.

Tzeng, F.-Y. & Ma, K.-L. (2005). Opening the black box - data driven visualization of neural networks. In *Proceedings of 16th IEEE Visualization Conference* (pp. 383–390). Minneapolis, MN, USA: IEEE Computer Society.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, 5(2), 241–259.

Yeh, I.-C. (1998). Modeling of strength of high performance concrete using artificial neural networks. *Cement and Concrete Research*, 28(12), 1797–1808.

Zakrzewski, R. R. (2001). Verification of a trained neural network accuracy. In *Proceedings of International Joint Conference on Neural Networks (IJCNN'01)*, volume 3 (pp. 1657–1662).: IEEE Press.